LEARNING FROM NATURAL HUMAN INTERACTIONS FOR ASSISTIVE ROBOTS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Ashesh Jain

May 2016

LEARNING FROM NATURAL HUMAN INTERACTIONS FOR ASSISTIVE ROBOTS

Ashesh Jain, Ph.D.

Cornell University 2016

Leveraging human knowledge to train robots is a core problem in robotics. In the near future we will see humans interacting with agents such as, assistive robots, cars, smart houses, etc. Agents that can elicit and learn from such interactions will find use in many applications. Previous works have proposed methods for learning low-level robotic controls or motion primitives from (near) optimal human signals. In many applications such signals are not naturally available. Furthermore, optimal human signals are also difficult to elicit from non-expert users at a large scale.

Understanding and learning user preferences from weak signals is therefore of great emphasis. To this end, in this dissertation we propose interactive learning systems which allow robots to learn by interacting with humans. We develop interaction methods that are natural to the end-user, and algorithms to learn from sub-optimal interactions. Furthermore, the interactions between humans and robots have complex spatio-temporal structure. Inspired by the recent success of powerful function approximators based on deep neural networks, we propose a generic framework for modeling interactions with structure of Recurrent Neural Networks. We demonstrate applications of our work on real-world scenarios on assistive robots and cars. This work also established state-of-the-art on several existing benchmarks.

BIOGRAPHICAL SKETCH

Ashesh Jain was born and brought up in Allahabad, India. Before joining the graduate program at Cornell University, he obtained a Bachelors degree in electrical engineering from the Indian Institute of Technology (IIT) Delhi. He has explored a variety of research areas: embedded systems, semiconductor device physics, machine learning, computer vision, and robotics – and he enjoyed all of them. In his free time, he likes to sing, cook, and swim.

This dissertation is dedicated to my parents – Renu Jain and Alok Jain.

ACKNOWLEDGEMENTS

I am very thankful to my adviser Ashutosh Saxena for his guidance. He always ensured that I address important research problems, and motivated me to think about the bigger picture. The rich experience of working with him will continue to guide me. I was also fortunate to collaborate with Thorsten Joachims. It helped me sketch the broader picture of this dissertation. He was very encouraging, and his research and teaching were truly inspirational. Professor Bart Selman was very supportive and encouraging throughout the PhD. He was always available for discussions, and ensured that I focused on research while he took care of the rest. I am thankful to Doug James and Bobby Kleinberg for sharing their deep insights whenever I presented my work. I also learned through various interactions with Clarie Cardie, Lillian Lee, and Percy Liang. I thank them for sharing their wisdom. A big thanks to the administrators, particularly Becky Stewart and Megan Gatch, who took care of almost everything.

I was fortunate to spend a significant part (two years) of my PhD at the Stanford AI Lab (SAIL). I am particularly thankful to Silvio Savarese and the CVGL group for hosting me. The research discussions with the students at SAIL were very enriching. I also thank Alex Sandra for helping with all the administrative work. The IT support at Stanford (Actions) was amazing and very efficient. I thank them for resolving my countless tickets. The time spent at Stanford was a thrilling experience. There I learned how research can translate into product!

I thank my lab members Yun Jiang, Hema Koppula, Ian Lenz, Jaeyong Sung, Dipendra Misra, Ozan Sener, and Chenxia Wu for the fun times and for helping me in various ways. I am also thankful to my wonderful collaborators: Hema Koppula, Amir Zamir, Ozan Sener, Dipendra Misra, Aditya Jami, Jayesh Gupta, Shane Soh, Bharad Raghavan, Avi Singh, Siddhant Manocha, Arpit Agarwal,

Debarghya Das, and Shikhar Sharma. I also learned through various research discussions with Adith Swaminathan, Moontae Lee, Karthik Raman, Anshumali Shrivastava, Ayush Dubey, Chen Wang, and Ozan Irsoy. My friends at Ithaca: Abhishek, Apoorv, Pankaj, Arzoo, Ayush, Aniket, Gauri, Chaitali, Chaitanya, and Avik, made this journey a lot more joyful.

Lastly, I am very thankful to my wonderful family – parents, wife, sisters, uncle, and aunt – for their love and support. My parents, Renu Jain and Alok Jain, supported me in every possible manner. Their love and affection kept me strong and going. I met my wife, Hema Koppula, during this period. We walked together the journey from friends to collaborators, and then becoming the partners-for-life. She was very supportive, and her advice always worked wonders for me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In our everyday lives, we interact with agents like personal computers, search engines, cars, etc., and reveal many of our personal choices, biases, and preferences. Improving agents by analyzing their interactions with humans is an active area of research. However, the algorithmic and systems challenges involved in interactive learning varies with the application. For example, there has been significant advances in learning from humans for information retrieval tasks such as recommendation systems [82, 126], ranking documents [102, 105, 199], etc. This is primarily due to the availability of scalable and easy-to-use interaction mechanisms (like search engines) that reap massive logs of interaction data.

As things stand in robotics, interactive learning is in its early stages due to the involved systems challenges, data scarcity, scaling difficulties, and the multi-modal nature of robotics problems. Nonetheless, learning from humans is a necessary step to unlock natural human-robot interactions with applications ranging from household robots to autonomous cars. Interactive human-robot learning presents new opportunities for developing novel interaction mechanisms, algorithms for sensory-fusion, and learning preferences grounded in the physical-world.

In this dissertation we study algorithms and applications of interactive learning between humans and robots. Figure 1.1 demonstrates two examples of interactions: one between a human and a household robot (Baxter) and another between a human driver and the car. In order for the agents (robot or car) to learn from such interactions, we address questions like:

Figure 1.1: **Robot and human interactions**

- *How do we design interaction mechanisms that are easy to use by non-expert users?*
- *How do we build learning algorithms that can learn from weak human signals?*
- *How do we model the rich spatio-temporal interactions using deep neural networks?*

1.1 Robots and humans: Interactive learning

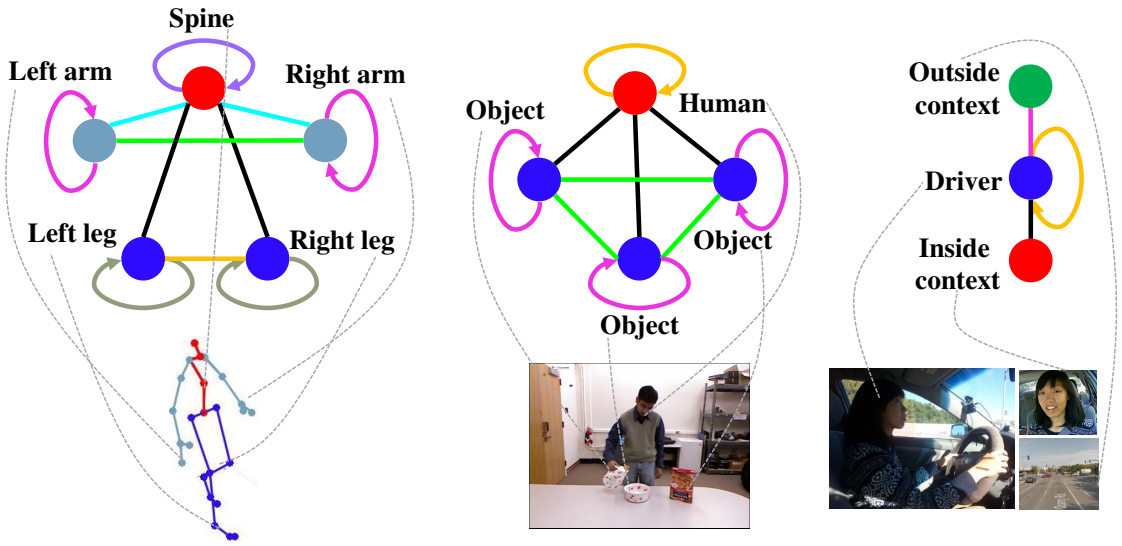
In order for robots to operate autonomously, they need to learn many concepts about our physical world. The skills they should acquire ranges from perception to performing actions in the human environment. While several skills can be learned from large annotated databases (e.g. object detection using Imagenet [46]), many others require the robot to observe and interact with humans. In this dissertation we focus on skills that robots learn by interacting with humans. This includes learning context dependent actions, understanding and anticipating human behaviour, and generating desirable motions to fulfill tasks.

The interaction mechanism between the human and the robot is also an important aspect to consider. First, the interaction being analyzed should vary based on the application and the nature of skill that the robot wants to learn. Second, it should also take into account the sub-optimality of human signals. In this dissertation we address several human-robot interactive learning problems and propose mechanisms for eliciting human signals. We focus on learning from non-expert users and build interaction mechanisms that are easy-to-use – *often as easy as a clicking a mouse* – to widely elicit user interaction. Figure 1.1 shows some examples of robots interacting with humans.

When interacting with robotic manipulators with high degrees-of-freedom, it becomes challenging for non-expert users to provide optimal kinesthetic demonstrations. In chapter 2 & 3, we consider such robotic manipulation and navigation tasks and learn user preferences over robot trajectories. We propose multiple easy-to-elicit feedback mechanisms and develop algorithms to learn from sub-optimal user feedback. We also develop a crowdsourcing platform to elicit such signals at a large scale.

We explore similar kinds of interactive learning for other agents. In chapter 4 we model the commonly occurring interaction between a driver, the car, and their surroundings. We propose a vehicular sensor-rich platform and learning algorithms to anticipate the future maneuvers of the driver several seconds in advance. In our interaction system, we equip a car with cameras, Global Positioning System (GPS), and a computing device. We observe many drivers driving to their destination, and the drivers provide many implicit examples of maneuvers as learning signals (lane changes, turns etc.).

In this dissertation we provide insights into developing such interactive



(a) Human motion modeling (b) Activity detection and anticipation (c) Maneuver anticipation

Figure 1.2: **Diverse spatio-temporal tasks and their corresponding spatio-temporal graphs**. The graph unrolls through time as the interaction progresses.

learning systems in order for robots to learn from humans. We particularly focus on interaction methods that are not just restricted to expert users but also generalize to non-experts.

1.2 Spatio-temporal deep structures

Many problems in robotics and computer vision involve complex interactions between components that span over space and time, such as interactions between a human, robot, and their surrounding environment. Figure 1.2 shows some examples of these spatio-temporal interactions. Modeling such spatio-temporal interactions and defining cost functions over them is central to learning from interactions. Spatio-temporal graphs are expressive tools for representing such high-level interactions. However, learning expressive cost func-

tions over them that capture long temporal dependencies is still a challenging problem.

Deep Recurrent Neural Networks (RNNs) have recently shown a lot of promise in the rich modeling of long-temporal sequences. Variants of RNN (e.g. RNNs with Long Short-Term Memory) further add external memory to the architecture. Despite their expressiveness, RNNs lack an intuitive high-level spatio-temporal structure. In chapter 5, we propose an approach for combining the power of high-level spatio-temporal graphs and sequence learning success of Recurrent Neural Networks (RNNs). We develop a scalable method for casting an arbitrary spatio-temporal graph as a rich RNN mixture that is feedforward, fully differentiable, and jointly trainable. The proposed method is generic and principled as it can be used for transforming any spatio-temporal graph by employing a certain set of well defined steps. We show applications of our approach by modeling several spatio-temporal interactions.

1.3 Sharing representations

The presence of organized knowledge bases has provided tremendous value to humans. Examples include the Google knowledge graph [50], IBM Watson [63], Wikipedia, and many others. By making the worlds knowledge available to everyone and everywhere, they have helped in the holistic growth of many areas of science and research. Unfortunately, the existing knowledge bases are of little use for robots who require much more fine-grained knowledge such as grasping, affordances, preferences, etc.

We address this problem more generally and introduce a knowledge engine

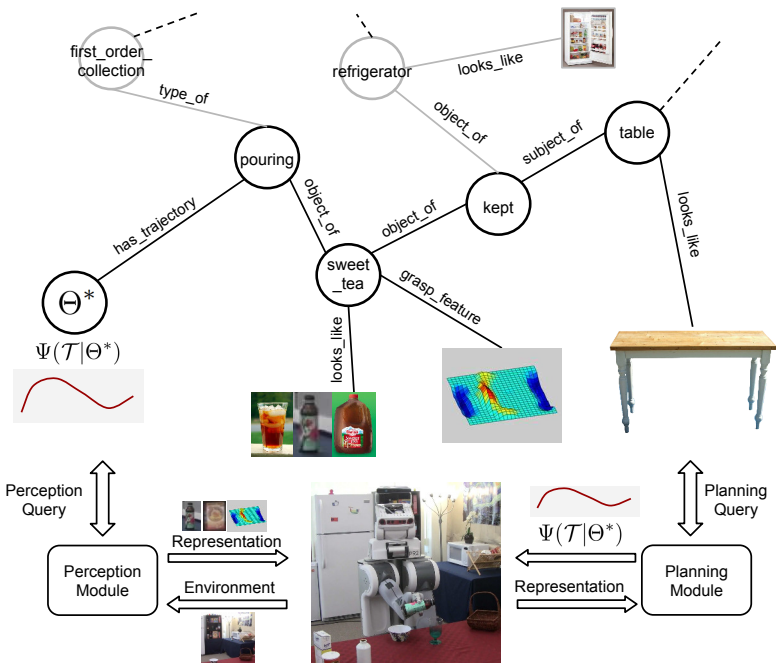
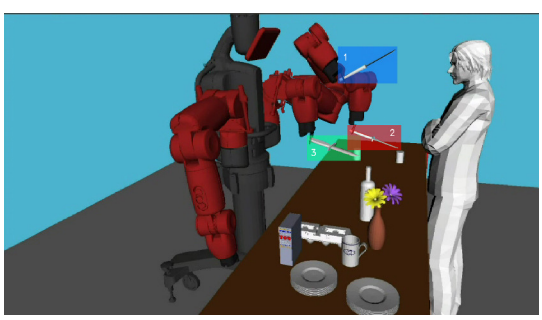


Figure 1.3: **An example use case of robot knowledge-engine** for performing tasks.

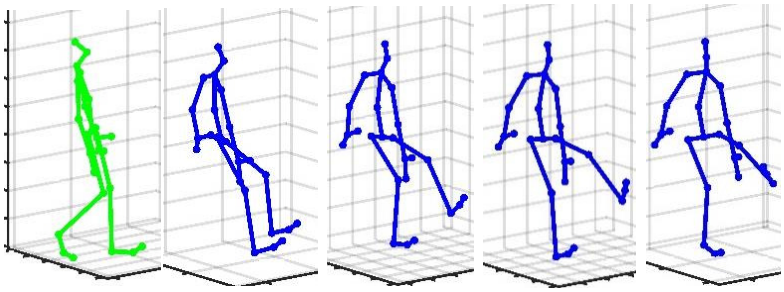
for robots in chapter 6. The knowledge engine allows robots to learn and share knowledge representations and enables them to carry out a variety of tasks (Figure 1.3). The knowledge stored in the engine comes from multiple sources including physical interactions that robots have while performing tasks (perception, planning and control), knowledge bases from the Internet, and learned representations from several robotics research groups. We propose a system architecture for storing the multi-modal knowledge base and demonstrate that knowledge sharing allows robots to generate better trajectories. This knowledge engine is developed in collaboration with several co-authors as a part of a bigger project[1].

[1]Saxena, Jain, Sener, Jami, Misra, Koppula `http://robobrain.me`



(a) Path planning

(b) Driving maneuver anticipation



(c) Human motion modeling

Figure 1.4: **Applications we address** by modeling different interactions.

1.4 Applications

We demonstrate the applications of our interactive learning algorithms on several real world robotic problems. Figure 1.4 shows some of the applications.

We address robotic navigation and manipulation tasks and learn user preferences over robot trajectories in contextually rich environments from suboptimal feedback (Figure 1.4(a)). We implement our algorithm on two high degrees-of-freedom robots, PR2 and Baxter, and apply it to perform household chores and grocery checkout tasks.

In order to anticipate driving maneuvers, we gather 1100 miles of driving data using our vehicular setup (Figure 1.4(b)). We represent the drivers interaction with the car and the surrounding environment over a spatio-temporal

graph and model the resulting graph with a deep RNN architecture. Our approach can anticipate maneuvers 3.5 seconds before they occur with high precision and recall. We also demonstrate the benefits of transforming spatio-temporal graphs to RNN architectures for applications like human motion modeling (Figure 1.4(c)) and human activity detection and anticipation.

1.5 First published appearances of the described contributions

Most of the contributions described in this dissertation have first appeared as the following publications.

- Chapter 2: Jain, Wojcik, Joachims, and Saxena [96]; Jain, Sharma, and Saxena [94]; Jain, Sharma, Joachims, and Saxena [93]
- Chapter 3: Jain, Das, Gupta, and Saxena [90]; Koppula, Jain, and Saxena [120]
- Chapter 4: Jain, Koppula, Raghavan, Soh, and Saxena [91]; Jain, Singh, Soh, Koppula, and Saxena [95]; Jain, Koppula, Soh, Raghavan, Singh, and Saxena [92]
- Chapter 5: Jain, Zamir, Savarese, and Saxena [97]
- Chapter 6: Saxena, Jain, Sener, Jami, Misra, and Koppula [194]

CHAPTER 2

LEARNING MANIPULATION TRAJECTORIES FROM ITERATIVE IMPROVEMENTS

Mobile manipulators with high degrees-of-freedom (DoF) can problem a task in many ways, however not all ways are desirable. In this chapter we consider the problem of learning preferences over trajectories for mobile manipulators such as personal robots and assembly line robots. The contributions of this chapter is a novel human-robot interaction framework for learning from non-expert users, and its applications to household and industrial robots.

The preferences we learn are more intricate than simple geometric constraints on trajectories; they are rather governed by the surrounding context of various objects and human interactions in the environment. We propose a coactive online learning framework for teaching preferences in contextually rich environments. The key novelty of our approach lies in the type of feedback expected from the user: the human user does not need to demonstrate optimal trajectories as training data, but merely needs to iteratively provide trajectories that slightly improve over the trajectory currently proposed by the system. We argue that this coactive preference feedback can be more easily elicited than demonstrations of optimal trajectories. Nevertheless, theoretical regret bounds of our algorithm match the asymptotic rates of optimal trajectory algorithms.

We implement our algorithm on two high DoF robots, PR2 and Baxter, and present three intuitive mechanisms for providing such incremental feedback. In our experimental evaluation we consider two context rich settings – household chores and grocery store checkout – and show that users are able to train the robot with just a few feedbacks (taking only a few minutes).

2.1 Beyond geometric path planning

Recent advances in robotics have resulted in mobile manipulators with high degree of freedom (DoF) arms. However, the use of high DoF arms has so far been largely successful only in structured environments such as manufacturing scenarios, where they perform repetitive motions (e.g., recent deployment of Baxter on assembly lines). One challenge in the deployment of these robots in unstructured environments (such as a grocery checkout counter or at our homes) is their lack of understanding of user preferences and thereby not producing desirable motions. In this chapter we address the problem of learning preferences over trajectories for high DoF robots such as Baxter or PR2. We consider a variety of household chores for PR2 and grocery checkout tasks for Baxter.

A key problem for high DoF manipulators lies in identifying an appropriate trajectory for a task. An appropriate trajectory not only needs to be valid from a geometric point (i.e., feasible and obstacle-free, the criteria that most path planners focus on), but it also needs to satisfy the user's preferences. Such users' preferences over trajectories can be common across users or they may vary between users, between tasks, and between the environments the trajectory is performed in. For example, a household robot should move a glass of water in an upright position without jerks while maintaining a safe distance from nearby electronic devices. In another example, a robot checking out a knife at a grocery store should strictly move it at a safe distance from nearby humans. Furthermore, straight-line trajectories in Euclidean space may no longer be the preferred ones. For example, trajectories of heavy items should not pass over fragile items but rather move around them. These preferences are often hard to describe and anticipate without knowing where and how the robot is deployed.

This makes it infeasible to manually encode them in existing path planners (e.g., Zucker et al. [256], Sucan et al. [209], Schulman et al. [195]) a priori.

In this chapter we propose an algorithm for learning user preferences over trajectories through interactive feedback from the user in a coactive learning setting [199]. In this setting the robot learns through iterations of user feedback. At each iteration robot receives a task and it predicts a trajectory. The user responds by slightly improving the trajectory but not necessarily revealing the optimal trajectory. The robot use this feedback from user to improve its predictions for future iterations. Unlike in other learning settings, where a human first demonstrates optimal trajectories for a task to the robot [7], our learning model does not rely on the user's ability to demonstrate optimal trajectories a priori. Instead, our learning algorithm explicitly guides the learning process and merely requires the user to incrementally improve the robot's trajectories, thereby learning preferences of user and not the expert. From these interactive improvements the robot learns a general model of the user's preferences in an online fashion. We realize this learning algorithm on PR2 and Baxter robots, and also leverage robot-specific design to allow users to easily give preference feedback.

Our experiments show that a robot trained using this approach can autonomously perform new tasks and if need be, only a small number of interactions are sufficient to tune the robot to the new task. Since the user does *not* have to demonstrate a (near) optimal trajectory to the robot, the feedback is easier to provide and more widely applicable. Nevertheless, it leads to an online learning algorithm with provable regret bounds that decay at the same rate as for optimal demonstration algorithms (eg. Ratliff et al. [182]).

In our empirical evaluation we learn preferences for two high DoF robots, PR2 and Baxter, on a variety of household and grocery checkout tasks respectively. We design expressive trajectory features and show how our algorithm learns preferences from online user feedback on a broad range of tasks for which object properties are of particular importance (e.g., manipulating sharp objects with humans in the vicinity). We extensively evaluate our approach on a set of 35 household and 16 grocery checkout tasks, both in batch experiments as well as through robotic experiments wherein users provide their preferences to the robot. Our results show that our system not only quickly learns good trajectories on individual tasks, but also generalizes well to tasks that the algorithm has not seen before. In summary, our key contributions are:

1. We present an approach for teaching robots which does not rely on experts' demonstrations but nevertheless gives strong theoretical guarantees.
2. We design a robotic system with multiple easy to elicit feedback mechanisms to improve the current trajectory.
3. We implement our algorithm on two robotic platforms, PR2 and Baxter, and support it with a user study.
4. We consider preferences that go beyond simple geometric criteria to capture object and human interactions.
5. We design expressive trajectory features to capture contextual information. These features might also find use in other robotic applications.

In the following section we discuss related works. In section 2.3 we describe our system and feedback mechanisms. Our learning algorithm and trajectory

features are discussed in sections 2.4 and 2.4, respectively. Section 2.6 gives our experiments and results. We discuss future research directions and conclude in section 2.7.

2.2 Previous works on learning to plan

Path planning is one of the key problems in robotics. Here, the objective is to find a collision free path from a start to goal location. Over the last decade many planning algorithms have been proposed, such as sampling based planners by Lavalle and Kuffner [137], and Karaman and Frazzoli [107], search based planners by Cohen et al. [37], trajectory optimizers by Schulman et al. [195], and Zucker et al. [256] and many more [108]. However, given the large space of possible trajectories in most robotic applications simply a collision free trajectory might not suffice, instead the trajectory should satisfy certain constraints and obey the end user preferences. Such preferences are often encoded as a cost which planners optimize [107, 195, 256]. We address the problem of learning a cost over trajectories for context-rich environments, and from sub-optimal feedback elicited from non-expert users. We now describe related work in various aspects of this problem.

Learning from Demonstration (LfD): Teaching a robot to produce desired motions has been a long standing goal and several approaches have been studied. In LfD an expert provides demonstrations of optimal trajectories and the robot tries to mimic the expert. Examples of LfD includes, autonomous helicopter flights [172], ball-in-a-cup game [116], planning 2-D paths [180, 181], etc. Such settings assume that kinesthetic demonstrations are intuitive to an end-user and

it is clear to an expert what constitutes a good trajectory. In many scenarios, especially involving high DoF manipulators, this is extremely challenging to do [2].[1] This is because the users have to give not only the end-effector's location at each time-step, but also the full configuration of the arm in a spatially and temporally consistent manner. In Ratliff et al. [182] the robot observes optimal user feedback but performs approximate inference. On the other hand, in our setting, the user never discloses the optimal trajectory or feedback, but instead, the robot learns preferences from sub-optimal suggestions for how the trajectory can be improved.

Noisy demonstrations and other forms of user feedback: Some later works in LfD provided ways for handling noisy demonstrations, under the assumption that demonstrations are either near optimal, as in Ziebart et al. [254], or locally optimal, as in Levine et al. [141]. Providing noisy demonstrations is different from providing relative preferences, which are biased and can be far from optimal. We compare with an algorithm for noisy LfD learning in our experiments. Wilson et al. [245] proposed a Bayesian framework for learning rewards of a Markov decision process via trajectory preference queries. Our approach advances over [245] and Calinon et. al. [29] in that we model user as a utility maximizing agent. Further, our score function theoretically converges to user's hidden function despite recieving sub-optimal feedback. In the past, various interactive methods (e.g. human gestures) [22, 208] have been employed to teach assembly line robots. However, these methods required the user to interactively show the complete sequence of actions, which the robot then remembered for future use. Recent works by Nikolaidis et al. [169, 170] in human-robot col-

[1]Consider the following analogy. In search engine results, it is much harder for the user to provide the best web-pages for each query, but it is easier to provide relative ranking on the search results by clicking.

laboration learns human preferences over a sequence of sub-tasks in assembly line manufacturing. However, these works are agnostic to the user preferences over robot's trajectories. Our work could complement theirs to achieve better human-robot collaboration.

Learning preferences over trajectories: User preferences over robot's trajectories have been studied in human-robot interaction. Sisbot et. al. [204, 203] and Mainprice et. al. [150] planned trajectories satisfying user specified preferences in form of constraints on the distance of the robot from the user, the visibility of the robot and the user's arm comfort. Dragan et. al. [55] used functional gradients [184] to optimize for legibility of robot trajectories. We differ from these in that we take a data driven approach and *learn* score functions reflecting user preferences from sub-optimal feedback.

Planning from a cost function: In many applications, the goal is to find a trajectory that optimizes a cost function. Several works build upon the sampling-based planner RRT [137] to optimize various cost heuristics [60, 42, 88]. Additive cost functions with Lipschitz continuity can be optimized using optimal planners such as RRT* [107]. Some approaches introduce sampling bias [140] to guide the sampling based planner. Recent trajectory optimizers such as CHOMP [184] and TrajOpt [195] provide optimization based approaches to finding optimal trajectory. Our work is complementary to these in that we learn a cost function while the above approaches optimize a cost.

Our work is also complementary to few works in path planning. Berenson et al. [18] and Phillips et al. [175] consider the problem of trajectories for high-dimensional manipulators. For computational reasons they create a database

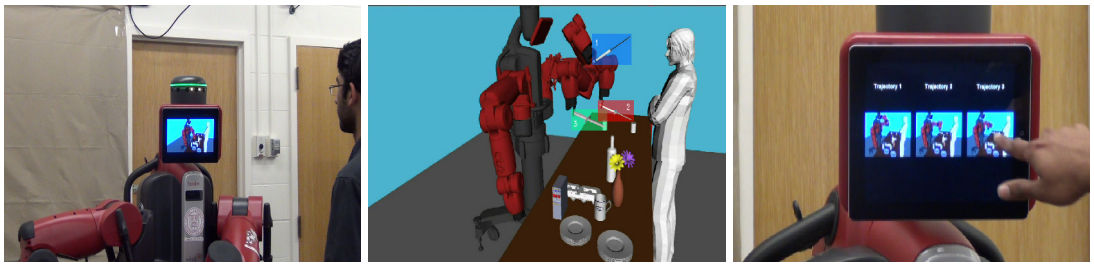


Figure 2.1: **Re-rank feedback mechanism: (Left)** Robot ranks trajectories using the score function and **(Middle)** displays top three trajectories on a touch screen device (iPad here). **(Right)** As feedback, the user improves the ranking by selecting the third trajectory.

of prior trajectories, which we could leverage to train our system. Other recent works consider generating human-like trajectories [53, 55, 216]. Humans-robot interaction is an important aspect and our approach could incorporate similar ideas.

Application domain: In addition to above mentioned differences we also differ in the applications we address. We capture the necessary contextual information for household and grocery store robots, while such context is absent in previous works. Our application scenario of learning trajectories for high DoF manipulations performing tasks in presence of different objects and environmental constraints goes beyond the application scenarios that previous works have considered. Some works in mobile robotics learn context-rich perception-driven cost functions, such as Silver et al. [201], Kretzschmar et al. [128] and Kitani et al. [113]. In this chapter we use features that consider robot configurations, object-object relations, and temporal behavior, and use them to learn a score function representing the preferences over trajectories.

2.3 Coactive learning with incremental feedback

We first give an overview of our robot learning setup and then describe in detail three mechanisms of user feedback.

2.3.1 Robot learning setup

We propose an online algorithm for learning preferences in trajectories from sub-optimal user feedback. At each step the robot receives a task as input and outputs a trajectory that maximizes its current estimate of some score function. It then observes user feedback – an improved trajectory – and updates the score function to better match the user preferences. This procedure of learning via iterative improvement is known as coactive learning. We implement the algorithm on PR2 and Baxter robots, both having two 7 DoF arms. In the process of training, the initial trajectory proposed by the robot can be far-off the desired behavior. Therefore, instead of directly executing trajectories in human environments, users first visualize them in the OpenRAVE simulator [48] and then decide the kind of feedback they would like to provide.

2.3.2 Feedback mechanisms

Our goal is to learn even from feedback given by non-expert users. We therefore require the feedback only to be *incrementally* better (as compared to being close to optimal) in expectation, and will show that such feedback is sufficient for the algorithm's convergence. This stands in contrast to learning from demonstra-

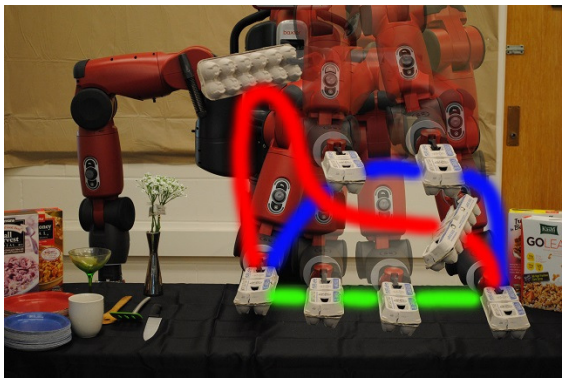


Figure 2.2: **Re-ranking feedback:** Shows three trajectories for moving egg carton from left to right. Using the current estimate of score function robot ranks them as red, green and blue. As feedback user clicks the green trajectory. **Preference**: Eggs are fragile. They should be kept upright and near the supporting surface.

tion (LfD) methods [172, 116, 180, 181] which require (near) optimal demonstrations of the complete trajectory. Such demonstrations can be extremely challenging and non-intuitive to provide for many high DoF manipulators [2]. Instead, we found [94, 96] that it is more intuitive for users to give incremental feedback on high DoF arms by improving upon a proposed trajectory. We now summarize three feedback mechanisms that enable the user to iteratively provide improved trajectories.

(a) Re-ranking: We display the ranking of trajectories using OpenRAVE [48] on a touch screen device and ask the user to identify whether any of the lower-ranked trajectories is better than the top-ranked one. The user sequentially observes the trajectories in order of their current predicted scores and clicks on the first trajectory which is better than the top ranked trajectory. Figure 2.1 shows three trajectories for moving a knife. As feedback, the user moves the trajectory at rank 3 to the top position. Likewise, Figure 2.2 shows three trajectories for moving an egg carton. Using the current estimate of score function robot ranks

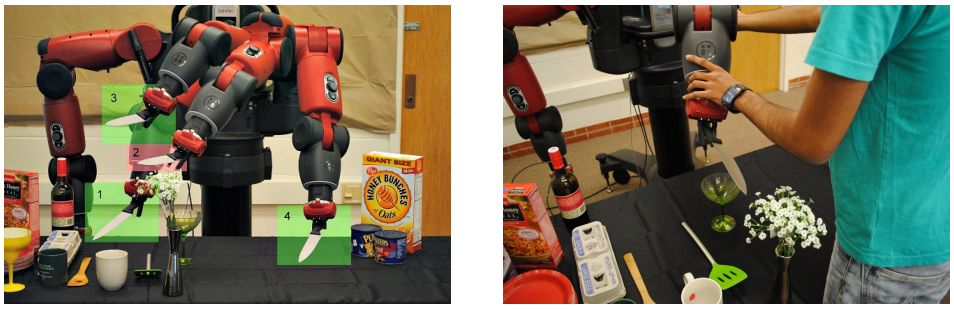


Figure 2.3: **Zero-G feedback:** Learning trajectory preferences from sub-optimal *zero-G* feedback. (**Left**) Robot plans a bad trajectory (waypoints 1-2-4) with knife close to flower. As feedback, user corrects waypoint 2 and moves it to waypoint 3. (**Right**) User providing *zero-G* feedback on waypoint 2.

them as red (1^{st}), green (2^{nd}) and blue (3^{rd}). Since eggs are fragile the user selects the green trajectory.

(b) Zero-G: This is a kinesthetic feedback. It allows the user to correct trajectory waypoints by physically changing the robot's arm configuration as shown in Figure 2.3. High DoF arms such as the Barrett WAM and Baxter have zero-force gravity-compensation (zero-G) mode, under which the robot's arms become light and the users can effortlessly steer them to a desired configuration. On Baxter, this zero-G mode is automatically activated when a user holds the robot's wrist (see Figure 2.3, right). We use this zero-G mode as a feedback method for incrementally improving the trajectory by correcting a waypoint. This feedback is useful (i) for bootstrapping the robot, (ii) for avoiding local maxima where the top trajectories in the ranked list are all bad but ordered correctly, and (iii) when the user is satisfied with the top ranked trajectory except for minor errors.

(c) Interactive: For the robots whose hardware does not permit zero-G feedback, such as PR2, we built an alternative interactive Rviz-ROS [74] interface for al-

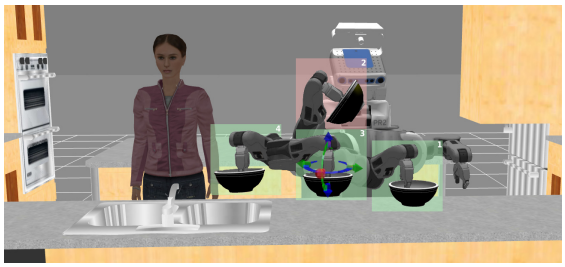


Figure 2.4: **Interactive feedback.** Task here is to move a bowl filled with water. The robot presents a bad trajectory with waypoints 1-2-4 to the user. As feedback user moves waypoint 2 (red) to waypoint 3 (green) using Rviz interactive markers. The interactive markers guides the user to correct the waypoint.

lowing the users to improve the trajectories by waypoint correction. Figure 2.4 shows a robot moving a bowl with one bad waypoint (in red), and the user provides a feedback by correcting it. This feedback serves the same purpose as zero-G.

Note that, in all three kinds of feedback the user never reveals the optimal trajectory to the algorithm, but only provides a slightly improved trajectory (in expectation).

2.4 Learning and feedback model

We model the learning problem in the following way. For a given task, the robot is given a context x that describes the environment, the objects, and any other input relevant to the problem. The robot has to figure out what is a good trajectory y for this context. Formally, we assume that the user has a scoring function $s^*(x, y)$ that reflects how much he values each trajectory y for context x. The higher the score, the better the trajectory. Note that this scoring function

cannot be observed directly, nor do we assume that the user can actually provide cardinal valuations according to this function. Instead, we merely assume that the user can provide us with *preferences* that reflect this scoring function. The robot's goal is to learn a function $s(x, y; w)$ (where w are the parameters to be learned) that approximates the user's true scoring function $s^*(x, y)$ as closely as possible.

Interaction Model. The learning process proceeds through the following repeated cycle of interactions.

- **Step 1:** The robot receives a context x and uses a planner to sample a set of trajectories, and ranks them according to its current approximate scoring function $s(x, y; w)$.
- **Step 2:** The user either lets the robot execute the top-ranked trajectory, or corrects the robot by providing an improved trajectory $\bar{y}$. This provides feedback indicating that $s^*(x, \bar{y}) > s^*(x, y)$.
- **Step 3:** The robot now updates the parameter w of $s(x, y; w)$ based on this preference feedback and returns to step 1.

Regret. The robot's performance will be measured in terms of regret, $REG_T = \frac{1}{T}\sum_{t=1}^{T}[s^*(x_t, y_t^*) - s^*(x_t, y_t)]$, which compares the robot's trajectory y_t at each time step t against the optimal trajectory y_t^* maximizing the user's unknown scoring function $s^*(x, y)$, $y_t^* = argmax_y s^*(x_t, y)$. Note that the regret is expressed in terms of the user's true scoring function s^*, even though this function is *never observed*. Regret characterizes the performance of the robot over its whole lifetime, therefore reflecting how well it performs *throughout* the learning process. We will employ learning algorithms with theoretical bounds on the regret for scoring

functions that are linear in their parameters, making only minimal assumptions about the difference in score between $s^*(x, \bar{y})$ and $s^*(x, y)$ in Step 2 of the learning process.

Expert Vs Non-expert user. We refer to an expert user as someone who can demonstrate the optimal trajectory y^* to the robot. For example, robotics experts such as, the pilot demonstrating helicopter maneuver in Abbeel et al. [172]. On the other hand, our non-expert users never demonstrate y^*. They can only provide feedback $\bar{y}$ indicating $s^*(x, \bar{y}) > s^*(x, y)$. For example, users working with assistive robots on assembly lines.

2.5 Our approach

For each task, we model the user's scoring function $s^*(x, y)$ with the following parametrized family of functions.

$$s(x, y; w) = w \cdot \phi(x, y) \tag{2.1}$$

w is a weight vector that needs to be learned, and $\phi(\cdot)$ are features describing trajectory y for context x. Such linear representation of score functions have been previously used for generating desired robot behaviors [172, 181, 254].

We further decompose the score function in two parts, one only concerned with the objects the trajectory is interacting with, and the other with the object being manipulated and the environment

$$\begin{aligned} s(x, y; w_O, w_E) &= s_O(x, y; w_O) + s_E(x, y; w_E) \\ &= w_O \cdot \phi_O(x, y) + w_E \cdot \phi_E(x, y) \end{aligned} \tag{2.2}$$

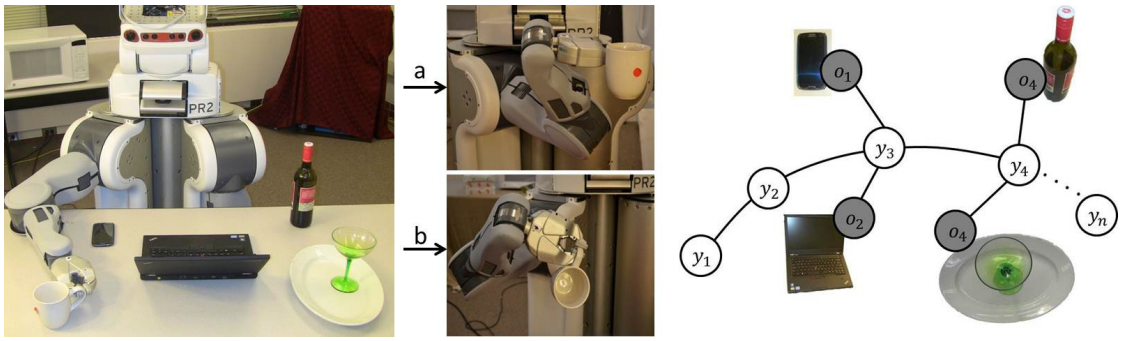


Figure 2.5: **(Left)** An environment with a few objects where the robot was asked to move the cup on the left to the right. **(Middle)** There are two ways of moving it, 'a' and 'b', both are suboptimal in that the arm is contorted in 'a' but it tilts the cup in 'b'. Given such constrained scenarios, we need to reason about such subtle preferences. **(Right)** We encode preferences concerned with object-object interactions in a score function expressed over a graph. Here $y_1, \ldots, y_n$ are different waypoints in a trajectory. The shaded nodes corresponds to environment (table node not shown here). Edges denotes interaction between nodes.

We now describe the features for the two terms, $\phi_O(\cdot)$ and $\phi_E(\cdot)$ in the following.

2.5.1 Features describing object-object interactions

This feature captures the interaction between objects in the environment with the object being manipulated. We enumerate waypoints of trajectory y as $y_1, .., y_N$ and objects in the environment as $O = \{o_1, .., o_K\}$. The robot manipulates the object $\bar{o} \in O$. A few of the trajectory waypoints would be affected by the other objects in the environment. For example in Figure 2.5, o_1 and o_2 affect the waypoint y_3 because of proximity. Specifically, we connect an object o_k to a trajectory waypoint if the minimum distance to collision is less than a threshold or if o_k lies

below $\bar{o}$. The edge connecting y_j and o_k is denoted as $(y_j, o_k) \in \mathcal{E}$.

Since it is the attributes [124] of the object that really matter in determining the trajectory quality, we represent each object with its *attributes*. Specifically, for every object o_k, we consider a vector of M binary variables $[l_k^1, .., l_k^M]$, with each $l_k^m = \{0, 1\}$ indicating whether object o_k possesses property m or not. For example, if the set of possible properties are {heavy, fragile, sharp, hot, liquid, electronic}, then a laptop and a glass table can have labels $[0, 1, 0, 0, 0, 1]$ and $[0, 1, 0, 0, 0, 0]$ respectively. The binary variables l_k^p and l^q indicates whether o_k and $\bar{o}$ possess property p and q respectively.[2] Then, for every (y_j, o_k) edge, we extract following four features $\phi_{oo}(y_j, o_k)$: projection of minimum distance to collision along x, y and z (vertical) axis and a binary variable, that is 1, if o_k lies vertically below $\bar{o}$, 0 otherwise.

We now define the score $s_O(\cdot)$ over this graph as follows:

$$s_O(x, y; w_O) = \sum_{(y_j, o_k) \in \mathcal{E}} \sum_{p,q=1}^{M} l_k^p l^q [w_{pq} \cdot \phi_{oo}(y_j, o_k)] \tag{2.3}$$

Here, the weight vector w_{pq} captures interaction between objects with properties p and q. We obtain w_O in eq. (2.2) by concatenating vectors w_{pq}. More formally, if the vector at position i of w_O is w_{uv} then the vector corresponding to position i of $\phi_O(x, y)$ will be $\sum_{(y_j, o_k) \in \mathcal{E}} l_k^u l^v [\phi_{oo}(y_j, o_k)]$.

[2] In this work, our goal is to relax the assumption of unbiased and close to optimal feedback. We therefore assume complete knowledge of the environment for our algorithm, and for the algorithms we compare against. In practice, such knowledge can be extracted using an object attribute labeling algorithms such as in [124].

2.5.2 Trajectory features

We now describe features, $\phi_E(x, y)$, obtained by performing operations on a set of waypoints. They comprise the following three types of the features:

Robot arm configurations

While a robot can reach the same operational space configuration for its wrist with different configurations of the arm, not all of them are preferred [249]. For example, the contorted way of holding the cup shown in Figure 2.5 may be fine at that time instant, but would present problems if our goal is to perform an activity with it, e.g. doing the pouring activity. Furthermore, humans like to anticipate robots move and to gain users' confidence, robot should produce predictable and legible robotic motions [55].

We compute features capturing robot's arm configuration using the location of its elbow and wrist, w.r.t. to its shoulder, in cylindrical coordinate system, (r, θ, z). We divide a trajectory into three parts in time and compute 9 features for each of the parts. These features encode the maximum and minimum r, θ and z values for wrist and elbow in that part of the trajectory, giving us 6 features. Since at the limits of the manipulator configuration, joint locks may happen, therefore we also add 3 features for the location of robot's elbow whenever the end-effector attains its maximum r, θ and z values respectively. Thus obtaining $\phi_{robot}(\cdot) \in \mathbb{R}^9$ (3+3+3=9) features for each one-third part and $\phi_{robot}(\cdot) \in \mathbb{R}^{27}$ for the complete trajectory.

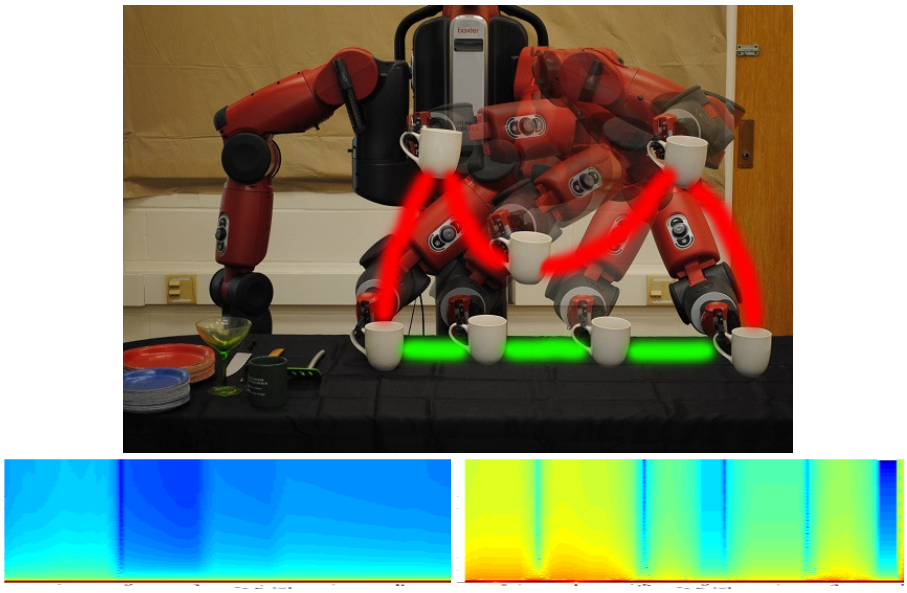


Figure 2.6: **(Top)** A good and bad trajectory for moving a mug. The bad trajectory undergoes ups-and-downs. **(Bottom)** Spectrograms for movement in z-direction: **(Left)** Good trajectory, **(Right)** Bad trajectory.

Orientation and temporal behaviour of the object to be manipulated

Object orientation during the trajectory is crucial in deciding its quality. For some tasks, the orientation must be strictly maintained (e.g., moving a cup full of coffee); and for some others, it may be necessary to change it in a particular fashion (e.g., pouring activity). Different parts of the trajectory may have different requirements over time. For example, in the placing task, we may need to bring the object closer to obstacles and be more careful.

We therefore divide trajectory into three parts in time. For each part we store the cosine of the object's maximum deviation, along the vertical axis, from its final orientation at the goal location. To capture object's oscillation along trajectory, we obtain a spectrogram for each one-third part for the movement of the object in x, y, z directions as well as for the deviation along vertical axis

(e.g. Figure 2.6). We then compute the average power spectral density in the low and high frequency part as eight additional features for each. This gives us 9 (=1+4*2) features for each one-third part. Together with one additional feature of object's maximum deviation along the whole trajectory, we get $\phi_{obj}(\cdot) \in \mathbb{R}^{28}$ (=9*3+1).

Object-environment interactions

This feature captures temporal variation of vertical and horizontal distances of the object $\bar{o}$ from its surrounding surfaces. In detail, we divide the trajectory into three equal parts, and for each part we compute object's: (i) minimum vertical distance from the nearest surface below it. (ii) minimum horizontal distance from the surrounding surfaces; and (iii) minimum distance from the table, on which the task is being performed, and (iv) minimum distance from the goal location. We also take an average, over all the waypoints, of the horizontal and vertical distances between the object and the nearest surfaces around it.[3] To capture temporal variation of object's distance from its surrounding we plot a time-frequency spectrogram of the object's vertical distance from the nearest surface below it, from which we extract six features by dividing it into grids. This feature is expressive enough to differentiate whether an object just grazes over table's edge (steep change in vertical distance) versus, it first goes up and over the table and then moves down (relatively smoother change). Thus, the features obtained from object-environment interaction are $\phi_{obj-env}(\cdot) \in \mathbb{R}^{20}$ (3*4+2+6=20).

Final feature vector is obtained by concatenating $\phi_{obj-env}$, ϕ_{obj} and ϕ_{robot}, giving us $\phi_E(\cdot) \in \mathbb{R}^{75}$.

[3]We query PQP collision checker plugin of OpenRave for these distances.

2.5.3 Computing trajectory rankings

For obtaining the top trajectory (or a top few) for a given task with context x, we would like to maximize the current scoring function $s(x, y; w_O, w_E)$.

$$y^* = \arg\max_y s(x, y; w_O, w_E). \tag{2.4}$$

Second, for a given set $\{y^{(1)}, \ldots, y^{(n)}\}$ of discrete trajectories, we need to compute (2.4). Fortunately, the latter problem is easy to solve and simply amounts to sorting the trajectories by their trajectory scores $s(x, y^{(i)}; w_O, w_E)$. Two effective ways of solving the former problem are either discretizing the state space [3, 21, 237] or directly sampling trajectories from the continuous space [19, 47]. Previously, both approaches have been studied. However, for high DoF manipulators the sampling based approach [19, 47] maintains tractability of the problem, hence we take this approach. More precisely, similar to [19], we sample trajectories using rapidly-exploring random trees (RRT) [137].[4] However, naively sampling trajectories could return many similar trajectories. To get diverse samples of trajectories we use various diversity introducing methods. For example, we introduce obstacles in the environment which forces the planner to sample different trajectories. Our methods also introduce randomness in planning by initizaling goal-sample bias of RRT planner randomly. To avoid sampling similar trajectories multiple times, one of our diversity method introduce obstacles to block waypoints of already sampled trajectories. Recent work by Ross et al. [188] propose the use of sub-modularity to achieve diversity. For more details on sampling trajectories we refer interested readers to the work by Erickson and LaValle [59], and Green and Kelly [77]. Since our primary goal is to learn a score

[4]When RRT becomes too slow, we switch to a more efficient bidirectional-RRT.The cost function (or its approximation) we learn can be fed to trajectory optimizers like CHOMP [184] or optimal planners like RRT* [107] to produce reasonably good trajectories.

Algorithm 1: Trajectory Preference Perceptron. (TPP)

Initialize $w_O^{(1)} \leftarrow 0$, $w_E^{(1)} \leftarrow 0$

for $t = 1$ to T **do**

Sample trajectories $\{y^{(1)}, ..., y^{(n)}\}$

$y_t = argmax_y s(x_t, y; w_O^{(t)}, w_E^{(t)})$

Obtain user feedback $\bar{y}_t$ (Re-ranking, Zero-G, or Interactive)

$w_O^{(t+1)} \leftarrow w_O^{(t)} + \phi_O(x_t, \bar{y}_t) - \phi_O(x_t, y_t)$

$w_E^{(t+1)} \leftarrow w_E^{(t)} + \phi_E(x_t, \bar{y}_t) - \phi_E(x_t, y_t)$

end for

function on trajectories we now describe our learning algorithm.

2.5.4 Learning the scoring function

The goal is to learn the parameters w_O and w_E of the scoring function $s(x, y; w_O, w_E)$ so that it can be used to rank trajectories according to the user's preferences. To do so, we adapt the Preference Perceptron algorithm [199] as detailed in Algorithm 1, and we call it the Trajectory Preference Perceptron (TPP). Given a context x_t, the top-ranked trajectory y_t under the current parameters w_O and w_E, and the user's feedback trajectory $\bar{y}_t$, the TPP updates the weights in the direction $\phi_O(x_t, \bar{y}_t) - \phi_O(x_t, y_t)$ and $\phi_E(x_t, \bar{y}_t) - \phi_E(x_t, y_t)$ respectively. Our update equation resembles to the weights update equation in Ratliff et al. [181]. However, our update does not depends on the availability of optimal demonstrations.

Despite its simplicity and even though the algorithm typically does not re-

ceive the optimal trajectory $y_t^* = \arg\max_y s^*(x_t, y)$ as feedback, the *TPP enjoys guarantees on the regret* [199]. We merely need to characterize by how much the feedback improves on the presented ranking using the following definition of expected α-informative feedback:

$$E_t[s^*(x_t, \bar{y}_t)] \geq s^*(x_t, y_t) + \alpha(s^*(x_t, y_t^*) - s^*(x_t, y_t)) - \xi_t$$

This definition states that the user feedback should have a score of $\bar{y}_t$ that is – in expectation over the users choices – higher than that of y_t by a fraction $\alpha \in (0, 1]$ of the maximum possible range $s^*(x_t, \bar{y}_t) - s^*(x_t, y_t)$. It is important to note that this condition only needs to be met in expectation and not deterministically. This leaves room for noisy and imperfect user feedback. If this condition is not fulfilled due to bias in the feedback, the slack variable ξ_t captures the amount of violation. In this way any feedback can be described by an appropriate combination of α and ξ_t. Using these two parameters, the proof by Shivaswamy and Joachims [199] can be adapted (for proof see Appendix A.1 & A.2) to show that average regret of TPP is upper bounded by:

$$E[REG_T] \leq O(\frac{1}{\alpha\sqrt{T}} + \frac{1}{\alpha T}\sum_{t=1}^{T}\xi_t)$$

In practice, over feedback iterations the quality of trajectory y proposed by robot improves. The α-informative criterion only requires the user to improve y to $\bar{y}$ in expectation.

2.6 Application on robots

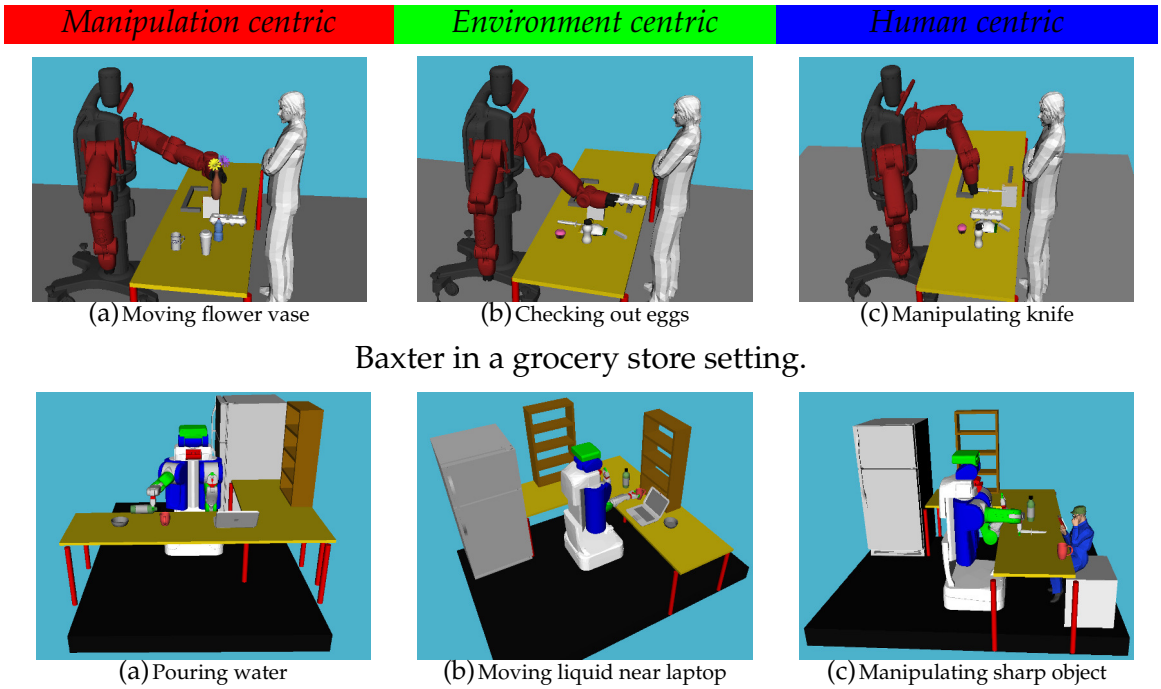
We first describe our experimental setup, then present quantitative results (Section 2.6.2) , and then present robotic experiments on PR2 and Baxter (Sec-

tion 2.6.4).

2.6.1 Experimental setup

Task and Activity Set for Evaluation. We evaluate our approach on 35 robotic tasks in a household setting and 16 pick-and-place tasks in a grocery store checkout setting. For household activities we use PR2, and for the grocery store setting we use Baxter. To assess the generalizability of our approach, for each task we train and test on scenarios with different objects being manipulated and/or with a different environment. We evaluate the quality of trajectories after the robot has grasped the item in question and while the robot moves it for task completion. Our work complements previous works on grasping items [193, 139], pick and place tasks [101], and detecting bar codes for grocery checkout [114]. We consider the following three most commonly occurring activities in household and grocery stores:

1. *Manipulation centric:* These activities are primarily concerned with the object being manipulated. Hence the object's properties and the way the robot moves it in the environment are more relevant. Examples of such household activities are pouring water into a cup or inserting pen into a pen holder, as in Figure 2.7 (Left). While in a grocery store, such activities could include moving a flower vase or moving fruits and vegetables, which could be damaged when dropped or pushed into other items. We consider *pick-and-place, pouring and inserting activities* with following objects: cup, bowl, bottle, pen, cereal box, flower vase, and tomato. Further, in every environment we place many objects, along with the object to be



(a) Moving flower vase (b) Checking out eggs (c) Manipulating knife

Baxter in a grocery store setting.

(a) Pouring water (b) Moving liquid near laptop (c) Manipulating sharp object

PR2 in a household setting.

Figure 2.7: Robot demonstrating different grocery store and household activities with various objects (**Left**) *Manipulation centric:* while pouring water the tilt angle of bottle must change in a particular manner, similarly a flower vase should be kept upright. (**Middle**) *Environment centric:* laptop is an electronic device so robot must carefully move water near it, similarly eggs are fragile and should not be lifted too high. (**Right**) *Human centric:* knife is sharp and interacts with nearby soft items and humans. It should strictly be kept at a safe distance from humans. (**Best viewed in color**)

manipulated, to restrict simple straight line trajectories.

2. *Environment centric:* These activities are also concerned with the interactions of the object being manipulated with the surrounding objects. Our object-object interaction features (Section 2.5.1) allow the algorithm to learn preferences on trajectories for moving fragile objects like egg cartons or moving liquid near electronic devices, as in Figure 2.7 (Middle). We consider moving fragile items like egg carton, heavy metal boxes near

a glass table, water near laptop and other electronic devices.

3. *Human centric:* Sudden movements by the robot put the human in danger of getting hurt. We consider activities where a robot manipulates sharp objects such as knife, as in Figure 2.7 (Right), moves a hot coffee cup or a bowl of water with a human in vicinity.

Experiment setting. Through experiments we will study:

- *Generalization*: Performance of robot on tasks that it has not seen before.
- *No demonstrations*: Comparison of TPP to algorithms that also learn in absence of expert's demonstrations.
- *Feedback:* Effectiveness of different kinds of user feedback in absence of expert's demonstrations.

Baseline algorithms. We evaluate algorithms that learn preferences from online feedback under two settings: (a) *untrained*, where the algorithms learn preferences for a new task from scratch without observing any previous feedback; (b) *pre-trained*, where the algorithms are pre-trained on other similar tasks, and then adapt to a new task. We compare the following algorithms:

- *Geometric*: The robot plans a path, independent of the task, using a Bi-directional RRT (BiRRT) [137] planner.
- *Manual*: The robot plans a path following certain manually coded preferences.
- *TPP*: Our algorithm, evaluated under both *untrained* and *pre-trained* settings.

- *MMP-online*: This is an online implementation of the Maximum Margin Planning (MMP) [181, 183] algorithm. MMP attempts to make an expert's trajectory better than any other trajectory by a margin. It can be interpreted as a special case of our algorithm with 1-informative i.e. optimal feedback. However, directly adapting MMP [181] to our experiments poses two challenges: (i) we do not have knowledge of the optimal trajectory; and (ii) the state space of the manipulator we consider is too large, discretizing which makes intractable to train MMP.

  To ensure a fair comparison, we follow the MMP algorithm from [181, 183] and train it under similar settings as TPP. Algorithm 2 shows our implementation of MMP-online. It is very similar to TPP (Algorithm 1) but with a different parameter update step. Since both algorithms only observe user feedback and not demonstrations, MMP-online treats each feedback as a proxy for optimal demonstration. At every iteration MMP-online trains a structural support vector machine (SSVM) [104] using all previous feedback as training examples, and use the learned weights to predict trajectory scores in the next iteration. Since the argmax operation is performed on a set of trajectories it remains tractable. We quantify closeness of trajectories by the L_2-norm of the difference in their feature representations, and choose the regularization parameter C for training SSVM in hindsight, giving an unfair advantage to MMP-online.

Evaluation metrics. In addition to performing a user study (Section 2.6.4), we also designed two datasets to quantitatively evaluate the performance of our online algorithm. We obtained experts labels on 1300 trajectories in a grocery setting and 2100 trajectories in a household setting. Labels were on the basis of subjective human preferences on a Likert scale of 1-5 (where 5 is the best). Note

Algorithm 2: MMP-online

Initialize $w_O^{(1)} \leftarrow 0$, $w_E^{(1)} \leftarrow 0, \mathcal{T} = \{\}$

for $t = 1$ to T **do**

Sample trajectories $\{y^{(1)}, ..., y^{(n)}\}$

$y_t = argmax_y s(x_t, y; w_O^{(t)}, w_E^{(t)})$

Obtain user feedback $\bar{y}_t$

$\mathcal{T} = \mathcal{T} \cup \{(x_t, \bar{y}_t)\}$

$w_O^{(t+1)}, w_E^{(t+1)}$ = Train-SSVM($\mathcal{T}$) (Joachims et al. [104])

end for

that these absolute ratings are never provided to our algorithms and are only used for the quantitative evaluation of different algorithms.

We evaluate performance of algorithms by measuring how well they rank trajectories, that is, trajectories with higher Likert score should be ranked higher. To quantify the quality of a ranked list of trajectories we report normalized discounted cumulative gain (nDCG) [152] — criterion popularly used in Information Retrieval for document ranking. In particular we report nDCG at positions 1 and 3, equation (2.6). While nDCG@1 is a suitable metric for autonomous robots that execute the top ranked trajectory (e.g., grocery checkout), nDCG@3 is suitable for scenarios where the robot is supervised by humans, (e.g., assembly lines). For a given ranked list of items (trajectories here) nDCG at position k is defined as:

$$DCG@k = \sum_{i=1}^{k} \frac{l_i}{\log_2(i+1)} \tag{2.5}$$

$$nDCG@k = \frac{DCG@k}{IDCG@k}, \tag{2.6}$$

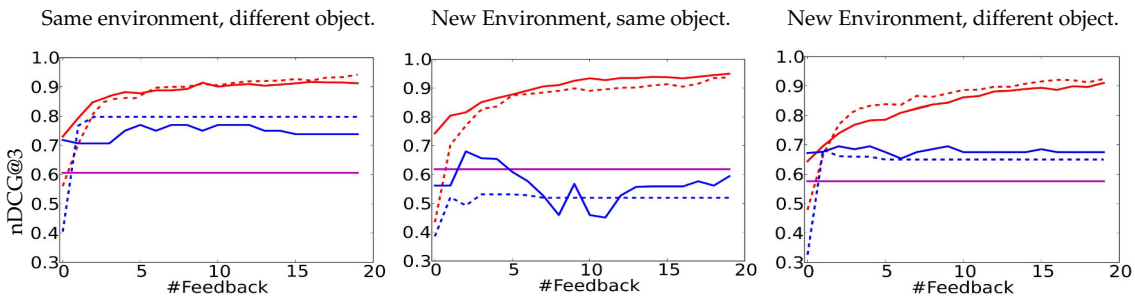


Figure 2.8: Results on Baxter in grocery store setting.

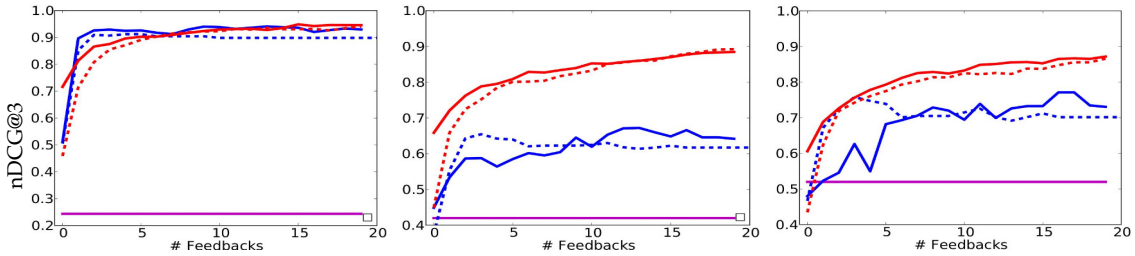


Figure 2.9: Results on PR2 in household setting.

Figure 2.10: Study of generalization with change in object, environment and both. Manual, Pre-trained MMP-online (—), Untrained MMP-online (– –), Pre-trained TPP (—), Untrained TPP (– –).

where l_i is the Likert score of the item at position i in the ranked list. *IDCG* is the *DCG* value of the best possible ranking of items. It is obtained by ranking items in decreasing order of their Likert score.

2.6.2 Results and discussion

We now present quantitative results where we compare TPP against the baseline algorithms on our data set of labeled trajectories.

How well does TPP generalize to new tasks? To study generalization of preference feedback we evaluate performance of TPP-pre-trained (i.e., *TPP* algorithm

under *pre-trained* setting) on a set of tasks the algorithm has not seen before. We study generalization when: (a) only the object being manipulated changes, e.g., a bowl replaced by a cup or an egg carton replaced by tomatoes, (b) only the surrounding environment changes, e.g., rearranging objects in the environment or changing the start location of tasks, and (c) when both change. Figure 2.10 shows nDCG@3 plots averaged over tasks for all types of activities for both household and grocery store settings.[5] TPP-pre-trained starts-off with higher nDCG@3 values than TPP-untrained in all three cases. However, as more feedback is provided, the performance of both algorithms improves, and they eventually give identical performance. We further observe that generalizing to tasks with both new environment and object is harder than when only one of them changes.

How does TPP compare to MMP-online? MMP-online while training assumes all user feedback is optimal, and hence over time it accumulates many contradictory/sub-optimal training examples. We empirically observe that MMP-online generalizes better in grocery store setting than the household setting (Figure 2.10), however under both settings its performance remains much lower than TPP. This also highlights the sensitivity of MMP to sub-optimal demonstrations.

How does TPP compare to Manual? For the manual baseline we encode some preferences into the planners, e.g., keep a glass of water upright. However, some preferences are difficult to specify, e.g., not to move heavy objects over fragile items. We empirically found (Figure 2.10) that the resultant manual algorithm produces poor trajectories in comparison with TPP, with an average nDCG@3 of 0.44 over all types of household activities. Table 2.1 reports

[5]Similar results were obtained with nDCG@1 metric.

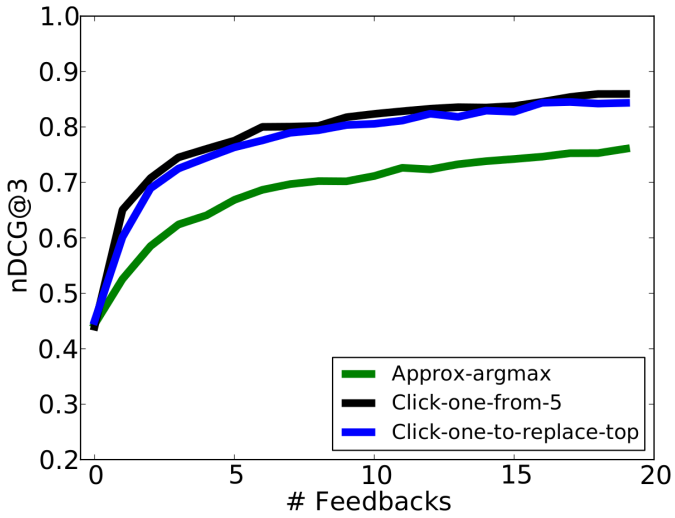


Figure 2.11: Study of re-rank feedback on Baxter for grocery store setting.

nDCG values averaged over 20 feedback iterations in untrained setting. For both household and grocery activities, TPP performs better than other baseline algorithms.

How does TPP perform with weaker feedback? To study the robustness of TPP to less informative feedback we consider the following variants of re-rank feedback:

- *Click-one-to-replace-top:* User observes the trajectories sequentially in order of their current predicted scores and clicks on the first trajectory which is better than the top ranked trajectory.
- *Click-one-from-5:* Top 5 trajectories are shown and user clicks on the one he thinks is the best after watching all 5 of them.
- *Approximate-argmax*: This is a weaker feedback, here instead of presenting top ranked trajectories, five random trajectories are selected as candidate. The user selects the best trajectory among these 5 candidates. This simulates a situation when computing an argmax over trajectories is prohibitive

Table 2.1: **Comparison of different algorithms in untrained setting.** Table contains nDCG@1(nDCG@3) values averaged over 20 feedbacks.

Algorithms	Grocery store setting on Baxter.				Household setting on PR2.			
	Manip. centric	Environ. centric	Human centric	Mean	Manip. centric	Environ. centric	Human centric	Mean
Geometric	.46 (.48)	.45 (.39)	.31 (.30)	.40 (.39)	.36 (.54)	.43 (.38)	.36 (.27)	.38 (.40)
Manual	.61 (.62)	.77 (.77)	.33 (.31)	.57 (.57)	.53 (.55)	.39 (.53)	.40 (.37)	.44 (.48)
MMP-online	.47 (.50)	.54 (.56)	.33 (.30)	.45 (.46)	.83 (.82)	.42 (.51)	.36 (.33)	.54 (.55)
TPP	**.88 (.84)**	**.90 (.85)**	**.90 (.80)**	**.89 (.83)**	**.93 (.92)**	**.85 (.75)**	**.78 (.66)**	**.85 (.78)**

and therefore approximated.

Figure 2.11 shows the performance of TPP-untrained receiving different kinds of feedback and averaged over three types of activities in grocery store setting. When feedback is more α-informative the algorithm requires fewer iterations to learn preferences. In particular, click-one-to-replace-top and click-one-from-5 are more informative than approximate-argmax and therefore require less feedback to reach a given nDCG@1 value. Approximate-argmax improves slowly since it is least informative. In all three cases the feedback is α-informative, for some $\alpha > 0$, therefore TPP-untrained eventually learns the user's preferences.

2.6.3 Comparison with fully-supervised algorithms

The algorithms discussed so far only observes ordinal feedback where the users iteratively improves upon the proposed trajectory. In this section we compare TPP to a fully-supervised algorithm that observes expert's labels while training. Eliciting such expert labels on the large space of trajectories is not realizable

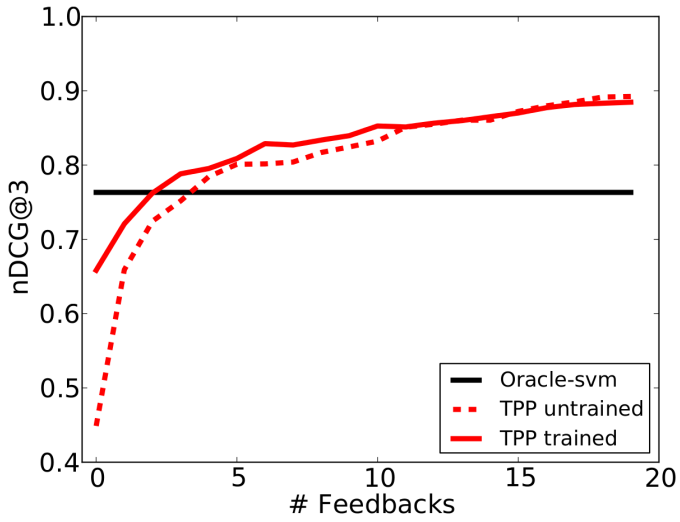


Figure 2.12: Comparision with fully-supervised Oracle-svm on Baxter for grocery store setting.

in practice. However, empirically it nonetheless provides an upper-bound on the generalization to new tasks. We refer to this algorithm as *Oracle-svm* and it learns to rank trajectories using SVM-rank [103]. Since expert labels are not available while prediction, on test set Oracle-svm predicts once and does not learn from user feedback.

Figure 2.12 compares TPP and Oracle-svm on new tasks. Without observing any feedback on new tasks Oracle-svm performs better than TPP. However, after few feedback iterations TPP improves over Oracle-svm, which is not updated since it requires expert's labels on test set. On average, we observe, it takes 5 feedback iterations for TPP to improve over Oracle-svm. Furthermore, learning from demonstration (LfD) can be seen as a special case of Oracle-svm where, instead of providing an expert label for every sampled trajectory, the expert directly demonstrates the optimal trajectory.

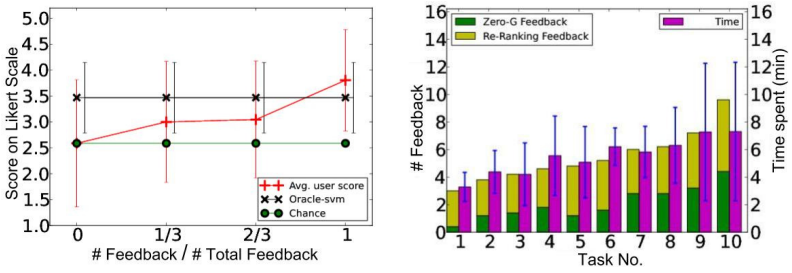


Figure 2.13: Grocery store setting on Baxter.

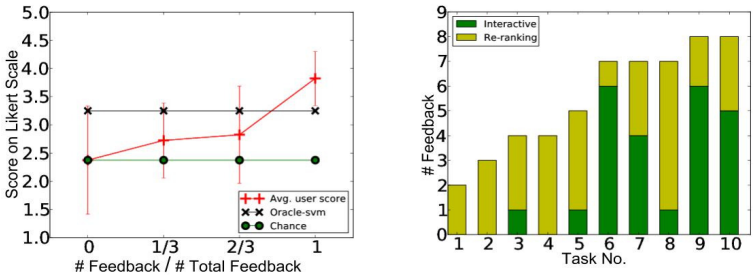


Figure 2.14: Household setting on PR2.

Figure 2.15: **(Left)** Average quality of the learned trajectory after every one-third of total feedback. **(Right)** Bar chart showing the average number of feedback (re-ranking and zero-G) and time required (only for grocery store setting) for each task. Task difficulty increases from 1 to 10.

2.6.4 Robotic experiment: User study in learning trajectories

We perform a user study of our system on Baxter and PR2 on a variety of tasks of varying difficulties in grocery store and household settings, respectively. Thereby we show a proof-of-concept of our approach in real world robotic scenarios, and that the combination of re-ranking and zero-G/interactive feedback allows users to train the robot in few feedback iterations.

Experiment setup: In this study, users not associated with this work, used our system to train PR2 and Baxter on household and grocery checkout tasks, re-

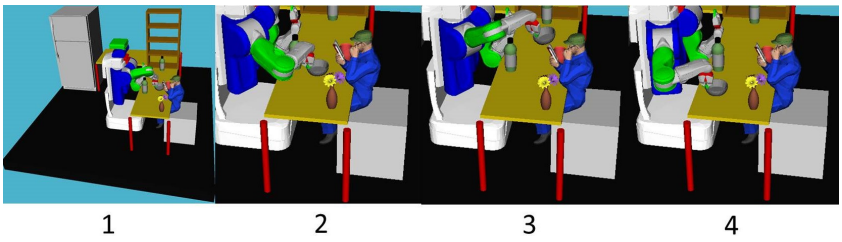


Figure 2.16: Shows trajectories for moving a bowl of water in presence of human. Without learning robot plans an undesirable trajectory and moves bowl over the human (waypoints 1-3-4). After six user feedback robot learns the desirable trajectory (waypoints 1-2-4).

spectively. Five users independently trained Baxter, by providing zero-G feedback kinesthetically on the robot, and re-rank feedback in a simulator. Two users participated in the study on PR2. On PR2, in place of zero-G, users provided interactive waypoint correction feedback in the Rviz simulator. The users were undergraduate students. Further, both users training PR2 on household tasks were familiar with Rviz-ROS.[6] A set of 10 tasks of varying difficulty level was presented to users one at a time, and they were instructed to provide feedback until they were satisfied with the top ranked trajectory. To quantify the quality of learning each user evaluated their own trajectories (self score), the trajectories learned by the other users (cross score), and those predicted by Oracle-svm, on a Likert scale of 1-5. We also recorded the total time a user spent on each task – from start of training till the user was satisfied with the top ranked trajectory. This includes time taken for both re-rank and zero-G feedback.

Is re-rank feedback easier to elicit from users than zero-G or interactive? In our user study, on average a user took 3 re-rank and 2 zero-G feedback per

[6]The smaller user size on PR2 is because it requires users with experience in Rviz-ROS. Further, we also observed users found it harder to correct trajectory waypoints in a simulator than providing zero-G feedback on the robot. For the same reason we report training time only on Baxter for grocery store setting.

Figure 2.17: Shows the learned trajectory for moving an egg carton. Since eggs are fragile robot moves the carton near the table surface. (Left) Start of trajectory. (Middle) Intermediate waypoint with egg close to the table surface. (Right) End of trajectory.

task to train a robot (Table 2.2). From this we conjecture, that for high DoF manipulators re-rank feedback is easier to provide than zero-G – which requires modifying the manipulator joint angles. However, an increase in the count of zero-G (interactive) feedback with task difficulty suggests (Figure 2.15 (Right)), users rely more on zero-G feedback for difficult tasks since it allows precisely rectifying erroneous waypoints. Figure 2.16 and Figure 2.17 show two example trajectories learned by a user.

How many feedback iterations a user takes to improve over Oracle-svm? Figure 2.15 (Left) shows that the quality of trajectory improves with feedback. On average, a user took 5 feedback to improve over Oracle-svm, which is also consistent with our quantitative analysis (Section 2.6.3). In grocery setting, users 4 and 5 were critical towards trajectories learned by Oracle-svm and gave them low scores. This indicates a possible mismatch in preferences between our expert (on whose labels Oracle-svm was trained) and users 4 and 5.

How do users' unobserved score functions vary? An average difference of 0.6 between users' self and cross score (Table 2.2) in the grocery checkout setting suggests preferences varied across users, but only marginally. In situations

Table 2.2: Shows learning statistics for each user. Self and cross scores of the final learned trajectories. The number inside bracket is standard deviation. **(Top)** Results for grocery store on Baxter. **(Bottom)** Household setting on PR2.

User	# Re-ranking feedback	# Zero-G feedback	Average time (min.)	Trajectory-Quality self	Trajectory-Quality cross
1	5.4 (4.1)	3.3 (3.4)	7.8 (4.9)	3.8 (0.6)	4.0 (1.4)
2	1.8 (1.0)	1.7 (1.3)	4.6 (1.7)	4.3 (1.2)	3.6 (1.2)
3	2.9 (0.8)	2.0 (2.0)	5.0 (2.9)	4.4 (0.7)	3.2 (1.2)
4	3.2 (2.0)	1.5 (0.9)	5.3 (1.9)	3.0 (1.2)	3.7 (1.0)
5	3.6 (1.0)	1.9 (2.1)	5.0 (2.3)	3.5 (1.3)	3.3 (0.6)

User	# Re-ranking feedbacks	# Interactive feedbacks	Trajectory-Quality self	Trajectory-Quality cross
1	3.1 (1.3)	2.4 (2.4)	3.5 (1.1)	3.6 (0.8)
2	2.3 (1.1)	1.8 (2.7)	4.1 (0.7)	4.1 (0.5)

where this difference is significant and a system is desired for a user population, a future work might explore coactive learning for satisfying user population, similar to Raman and Joachims [179]. For household setting, the sample size is small to draw a similar conclusion.

How long does it take for users to train a robot? We report training time for only the grocery store setting, because the interactive feedback in the household setting requires users with experience in Rviz-ROS. Further, we observed that users found it difficult to modify the robot's joint angles in a simulator to their desired configuration. In the grocery checkout setting, among all the users, user 1 had the strictest preferences and also experienced some early

difficulties in using the system and therefore took longer than others. On an average, a user took 5.5 minutes per task, which we believe is acceptable for most applications. Future research in human-computer interaction, visualization and better user interfaces [200] could further reduce this time. For example, simultaneous visualization of top ranked trajectories instead of sequentially showing them to users (the scheme we currently adopt) could bring down the time for re-rank feedback. Despite its limited size, through our user study we show that our algorithm is realizable in practice on high DoF manipulators. We hope this motivates researchers to build robotic systems capable of learning from *non-expert* users. For more details, videos and code, visit: `http://pr.cs.cornell.edu/coactive/`

2.7 Conclusion

When manipulating objects in human environments, it is important for robots to plan motions that follow users' preferences. In this chapter we considered preferences that go beyond simple geometric constraints and that considered surrounding context of various objects and humans in the environment. We presented a coactive learning approach for teaching robots these preferences through iterative improvements from non-expert users. Unlike in standard learning from demonstration approaches, our approach does not require the user to provide optimal trajectories as training data. We evaluated our approach on various household (with PR2) and grocery store checkout (with Baxter) settings. Our experiments suggest that it is indeed possible to train robots within a few minutes with just a few incremental feedbacks from non-expert users.

Future research could extend coactive learning to situations with uncertainty in object pose and attributes. Under uncertainty the trajectory preference perceptron will admit a belief space update form, and theoretical guarantees will also be different. Coactive feedback might also find use in other interesting robotic applications such as assistive cars, where a car learns from humans steering actions. Scaling up feedback by crowd-sourcing and exploring other forms of easy-to-elicit learning signals are also potential future directions.

CHAPTER 3

CROWDSOURCING FEEDBACK FOR PATH PLANNING

The feedback methods like zero-G, interactive feedback, or re-ranking feedback are easier to elicit than optimal demonstrations. However, despite their ease of use, they are difficult to elicit at large scale due to involved physical interactions between the human and robot. Leveraging robot simulators such as OpenRAVE [48], Gazebo [117] etc., together with the crowd, through platforms like Amazon Mechanical Turk, is an attractive alternate to gather massive amounts of weakly labeled data. Since the crowd are not experts, the interactions need to be accordingly designed.

In this chapter we design a crowdsourcing platform – PlanIt, for learning preferences over trajectories. In order to keep the user interaction simple, on PlanIt we show videos of robot moving in human environments. As feedback users simply reveal their like or dislike over parts of the video, but do not reveal their fine-grained reasoning behind the feedback. This allows us to collect a large amount of user feedback over many environments. We take a generative approach with latent variables for modeling the user feedback, and using the weak and noisy labels from PlanIt we learn the parameters of our model. We test our approach on 112 different environments for robotic navigation tasks. Our experiments show that the learned cost function generates preferred trajectories in human environments. Our crowdsourcing system is publicly available for the visualization of the learned costs and for providing preference feedback: `http://planit.cs.cornell.edu`

3.1 Planning affordances

One key problem robots face in performing tasks in human environments is identifying trajectories desirable to the users. In this chapter we present a crowdsourcing system PlanIt that learns user preferences by taking their feedback over the Internet. In previous works, user preferences are usually encoded as a cost over trajectories, and then optimized using planners such as RRT* [107], CHOMP [184], TrajOpt [195]. However, most of these works optimize expert-designed cost functions based on different geometric and safety criteria [204, 202, 150]. While satisfying safety criteria is necessary, they alone ignore the contextual interactions in human environments [94]. We take a data driven approach and learn a context-rich cost over the trajectories from the preferences shown by *non-expert* users.

In this chapter we model user preferences arising during human activities. Humans constantly engage in activities with their surroundings – watching TV or listening to music, etc. – during which they prefer minimal interruption from external agents that share their environment. For example, a robot that blocks the view of a human watching TV is not a desirable social agent. *How can a robot learn such preferences and context?* This problem is further challenging because human environments are unstructured, and as shown in Figure 3.1 an environment can have multiple human activities happening simultaneously. Therefore generalizing the learned model to new environments is a key challenge.

We formulate the problem as learning to ground each human activity to a spatial distribution signifying regions crucial to the activity. We refer to these spatial distributions as *planning affordances*[1] and parameterize the cost function

[1]Gibson [70] defined object affordances as possible actions that an agent can perform in an

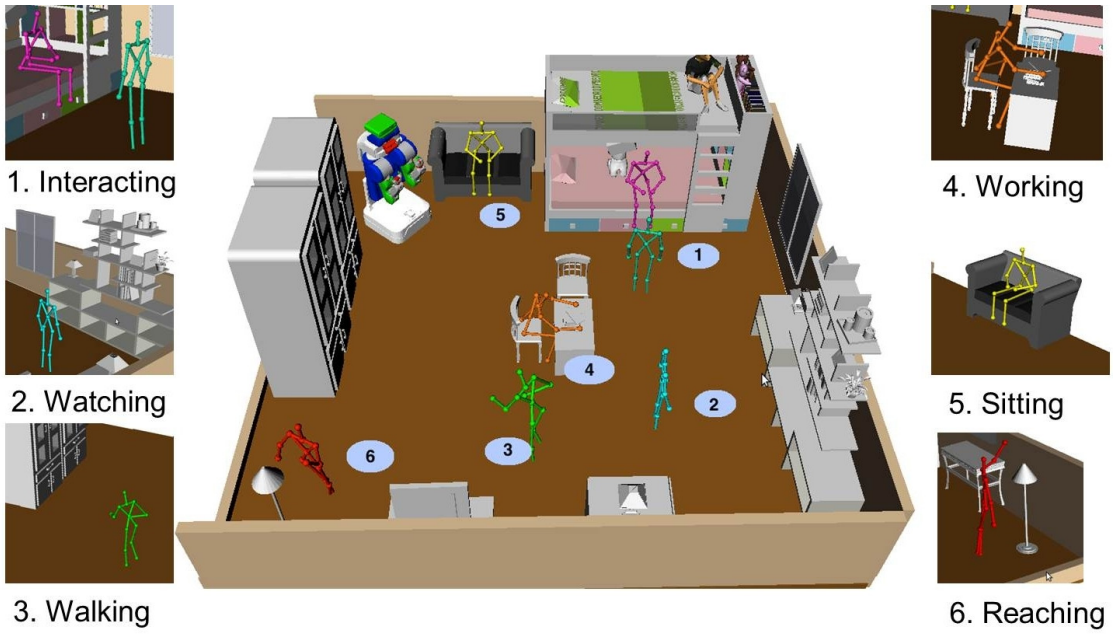


Figure 3.1: **Various human activities with the objects in the environment** affect how a robot should navigate in the environment. The figure shows an environment with multiple human activities: (1) two humans *interacting*, (2) *watching*, (3) *walking*, (4) *working*, (5) *sitting*, and (6) *reaching* for a lamp. We learn a spatial distribution for each activity, and use it to build a cost map (aka planning affordance map) for the complete environment. Using the cost map, the robot plans a preferred trajectory in the environment.

using these distributions. Our affordance representation is different by relating to the object's functionality, unlike previous works which have an object centric view. The commonly studied discrete representation of affordances [58, 161, 119, 111] are of limited use in planning trajectories. For example, a TV has a *watchable* affordance and undergoes a *watching* activity, however these labels themselves are not informative enough to convey to the robot that it should not move between the user and the TV. The grounded representation we propose is more useful for planning tasks than the discrete representations.

environment.

To generalize well across diverse environments we develop a crowdsourcing web-service `PlanIt` to collect large-scale preference data. On PlanIt we show short videos (mostly < 15 sec) to non-expert users of the robot navigating in context-rich environments with humans performing activities. As feedback users label segments of the videos as good, bad or neutral. While previous methods of eliciting feedback required expensive expert demonstrations in limited environments, PlanIt is usable by *non-expert* users and scales to a large number of environments. This simplicity comes at the cost of weak and noisy feedback. We present a generative model of the preference data obtained.

We evaluate our approach on a total of 112 bedroom and living room environments. We use OpenRave [48] to generate trajectories in these environments and upload them to PlanIt database for user feedback. We quantitatively evaluate our learned model and compare it to previous works on human-aware planning. Further, we validate our model on the PR2 robot to navigate in human environments. The results show that our learned model generalizes well to the environments not seen before.

In the following sections, we formally state the planning problem, give an overview of the PlanIt engine in Section 3.4, discuss the cost parametrization through affordance in Section 3.5.1, describe the learning algorithm in Section 3.5.2, and show the experimental evaluation in Section 3.6.

3.2 Previous works on affordances and learning preferences

Learning from demonstration (LfD). One approach to learning preferences is to mimic an expert's demonstrations. Several works have built on this idea such

as the autonomous helicopter flights [172], the ball-in-a-cup experiment [116], planning 2-D paths [181], etc. These approaches are applicable in our setting. However, they are expensive in that they require an expert to demonstrate the optimal trajectory. Such demonstrations are difficult to elicit on a large scale and over many environments. Instead we learn with preference data from non-expert users across a wide variety of environments.

Planning from a cost function. In many applications, the goal is to find a trajectory that optimizes a cost function. Several works build upon the sampling based planner RRT [137, 107] to optimize various cost heuristics [42, 88]. Some approaches introduce sampling bias [140] to guide the planner. Alternative approaches include recent trajectory optimizers CHOMP [184] and TrajOpt [195]. We are complementary to these works in that we learn a cost function while the above approaches optimize cost functions.

Modeling human motion for navigation path. Sharing environment with humans requires robots to model and predict human navigation patterns and generate socially compliant paths [17, 130, 255]. Recent works [121, 149, 244] model human motion to anticipate their actions for better human-robot collaboration. Instead we model the spatial distribution of human activities and the preferences associated with those activities.

Affordances in robotics. Many works in robotics have studied affordances. Most of the works study affordance as cause-effect relations, i.e. the effects of robot's actions on objects [58, 231, 233, 161]. We differ from these works in the representation of affordance and in its application to planning user preferred trajectories. Further, we consider context-rich environments where humans interact with various objects, while such context was not important to previous

works. Similar to Jiang et al. [100], our affordances are also distributions, but they used them for scene arrangement while we use them for planning.

User preferences in path planning. User preferences have been studied in human-robot interaction literature. Sisbot et al. [204, 203] and Mainprice et al. [150] planned trajectories satisfying user specified preferences such as the distance of the robot from humans, visibility of the robot and human arm comfort. Dragan et al. [53] used functional gradients [184] to optimize for legibility of robot trajectories. We differ from these in that we *learn* the cost function capturing preferences arising during human-object interactions. Jain et al. [96, 94] learned a context-rich cost via iterative feedback from non-expert users. Similarly, we also learn from the preference data of non-expert users. However, we use crowdsourcing like Chung et al. [36] for eliciting user feedback which allows us to learn from large amount of preference data. In experiments, we compare against Jain's trajectory preference perceptron algorithm.

3.3 Context-aware planning problem

The planning problem we address is: given a goal configuration G and a context-rich environment E (containing objects, humans and activities), the algorithm should output a desirable trajectory $\hat{\mathcal{T}}$. We consider navigation trajectories and represent them as a sequence of discrete 2D waypoints, i.e., $\mathcal{T} = \{t_1, \ldots, t_n\}$.

In order to encode the user's desirability we use a positive cost function $\mathbf{\Psi}(\cdot)$ that maps trajectories to a scalar value. Trajectories with lower cost indicate greater desirability. We denote the cost of trajectory $\mathcal{T}$ in environment E as

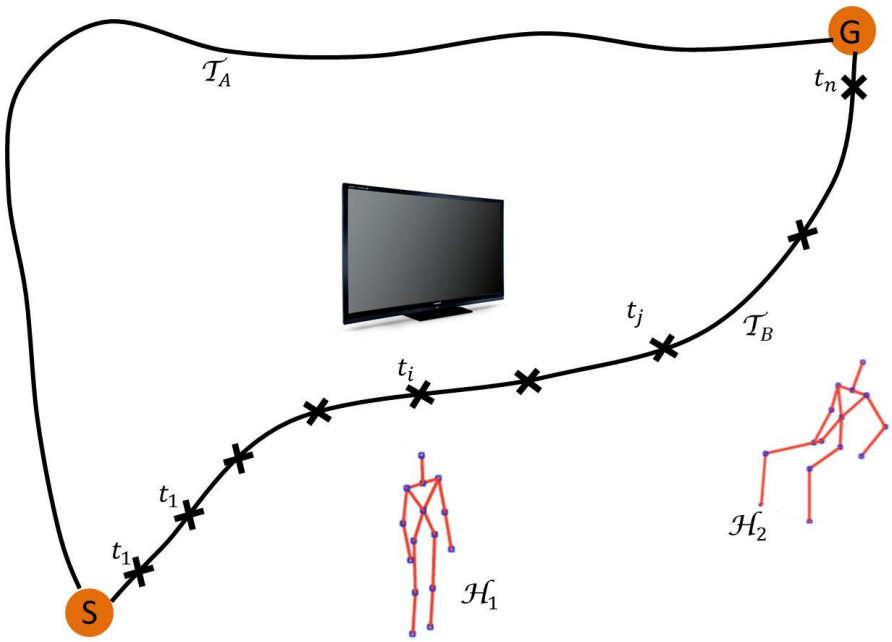


Figure 3.2: **Preference-based Cost calculation of a trajectory.** The trajectory $\mathcal{T}_A$ is preferred over $\mathcal{T}_B$ because it does not interfere with the human activities. The cost of a trajectory decomposes over the waypoints t_i, and the cost depends on the location of the objects and humans associated with an activity.

$\mathbf{\Psi}(\mathcal{T}|E)$ where $\mathbf{\Psi}$ is defined as:

$$\mathbf{\Psi}|E : \mathcal{T} \longrightarrow \mathbb{R}$$

The context-rich environment E comprises humans, objects and activities. Specifically, it models the human-human and human-object interactions. The robot's goal is to learn the spatial distribution of these interactions in order to plan good trajectories that minimally interrupt human activities. The key challenge here lies in designing an expressive cost function that accurately reflects user preferences, captures the rich environment context, and can be learned from data.

Figure 3.2 illustrates how the cost of a trajectory is the cumulative effect of

the environment at each waypoint. We thus define a trajectory's cost as a product of the costs over each waypoint:

$$\mathbf{\Psi}(\mathcal{T} = \{t_1, .., t_n\}|E) = \prod_i \Psi_{a_i}(t_i|E) \tag{3.1}$$

In the above equation, $\Psi_{a_i}(t_i|E)$ is the cost of waypoint t_i and its always positive.[2] Because user preferences vary over activities, we learn a separate cost for each activity. $\Psi_a(\cdot)$ denotes the cost associated with an activity $a \in E$. The robot navigating along a trajectory often interferes with multiple human activities e.g., trajectory $\mathcal{T}_\mathcal{B}$ in Figure 3.2. Thus we associate with each waypoint t_i an activity a_i it interacts with, as illustrated in Eq. (3.1).

The cost function changes with the activities happening in the environment. As illustrated in Figure 3.2, the robot prefers the trajectory $\mathcal{T}_A$ over the otherwise preferred shorter trajectory $\mathcal{T}_B$ because the latter interferes with human interactions (e.g., H_2 is watching TV).

3.4 PlanIt: A crowdsourcing engine

Rich data along with principled learning algorithms have achieved much success in robotics problems such as grasping [41, 157, 173], manipulation [110], trajectory modeling [237] etc. Inspired by such previous works, we design *PlanIt*: a scalable approach for learning user preferences over robot trajectories across a wide-variety of environments: `http://planit.cs.cornell.edu`

On PlanIt's webpage users watch videos of robot navigating in contextually-

[2]Since the cost is always positive, the product of costs in Equation (3.1) is equivalent to the sum of logarithmic cost.

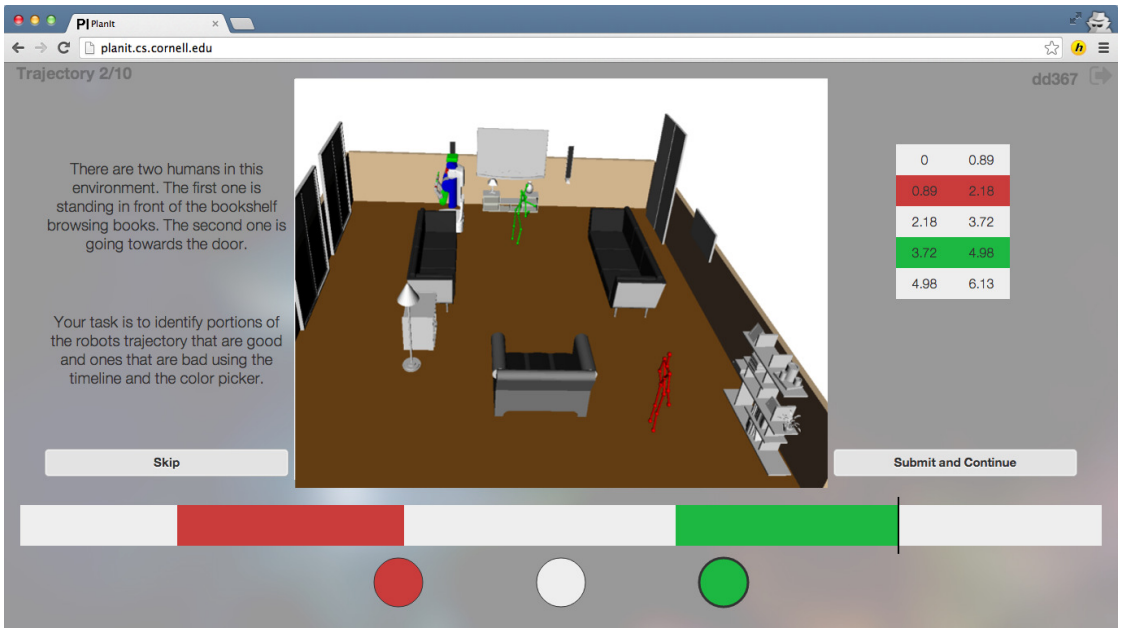


Figure 3.3: **PlanIt Interface.** Screenshot of the PlanIt video labeling interface. The video shows a human walking towards the door while other human is browsing books, with text describing the environment on left. As feedback the user labels the time interval where the robot crosses the human browsing books as red, and the interval where the robot carefully avoids the walking human as green. (Best viewed in color)

rich environments and reveal their preferences by labeling video segments (Figure 3.3). We keep the process simple for users by providing three label choices $\{bad, neutral, good\}$. For example, the trajectory segments where the robot passes between a human and TV can be labeled as bad, and segments where it navigates in open space as neutral. We now discuss three aspects of PlanIt.

A. Weak labels from PlanIt: In PlanIt's feedback process, users only label parts of a trajectory (i.e. sub-trajectory) as good, bad or neutral. For the ease of usability and to reduce the labeling effort, users only provide the labels and do not reveal the (*latent*) reason for the labels. We capture the user's intention as a latent variable in the learning algorithm (discussed in Section 3.5.2).

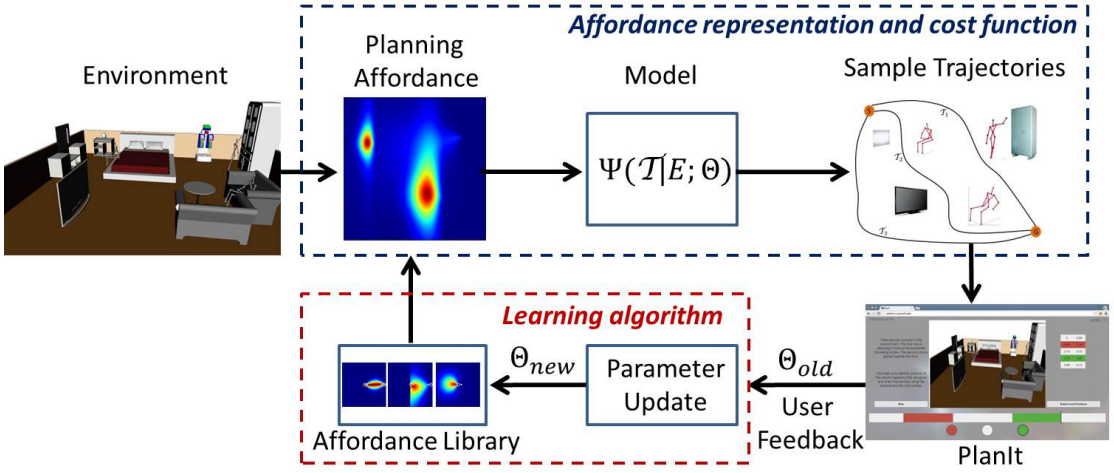


Figure 3.4: **An illustration of our PlanIt system.** Our learning system has three main components (i) cost parameterization through affordance; (ii) The PlanIt engine for receiving user preference feedback; and (iii) Learning algorithm. (Best viewed in color)

The user feedback in PlanIt is in contrast to other learning-based approaches such as learning from the expert's demonstrations (LfD) [172, 116, 181, 2] or the co-active feedback [96, 94]. In both LfD and co-active learning approaches it is time consuming and expensive to collect the preference data on a robotic platform and across many environments. Hence these approaches learn using limited preference data from users. On the other hand, PlanIt's main objective is to leverage the crowd and learn from the *non-expert* users across a large number of environments.

B. Generating robot trajectory videos: We sample many trajectories (using RRT [137]) for the PR2 robot in human environments using OpenRAVE [48]. We video record these trajectories and add them to PlanIt's trajectory database. The users watch the short videos of the PR2 interacting with human activities, and reveal their preferences. We also ensure that trajectories in the database are diverse by following the ideas presented in [19, 96]. As of now the PlantIt's

database has 2500 trajectories over 112 environments. In Section 3.6 we describe the data set.

C. Learning system: In our learning system, illustrated in Figure 3.4, the learned model improves as more preference data from users become available. We maintain an affordance library with spatial distributions for each human activity. When the robot observes an environment, it uses the distributions from the library and builds a planning affordance map (aka cost function) for the environment. The robot then samples trajectories from the cost function and presents them on the PlanIt engine for feedback.

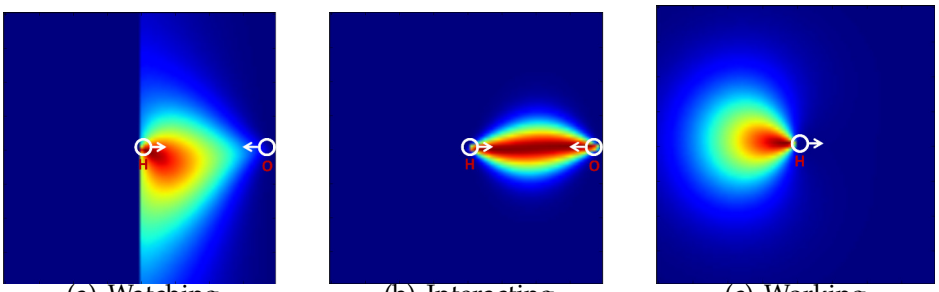
3.5 Learning algorithm

We first discuss our parameterization of the cost function and then the procedure for learning the model parameters.

3.5.1 Cost parameterization through affordance

In order to plan trajectories in human environments we model the human-object relationships. These relationships are called 'object affordances'. In this chapter we model the affordances such that they are relevant to path planning and we refer to them as 'planning affordances'.

Specifically, we learn the spatial distribution of the human-object interactions. For example, a TV has a *watchable* affordance and therefore the space between the human and the TV is relevant for the *watching* activity. Since a



(a) Watching (b) Interacting (c) Working

Figure 3.5: **An example the learned planning affordance.** In the top-view, the human is at the center and facing the object on the right (1m away). Dimension is 2m×2m. (Best viewed in color)

watchable label by itself is not informative enough to help in planning we ground it to a spatial distribution. Figure 3.5(a) illustrates the learned spatial distribution when the human watches TV. Similarly, a chair is *sittable* and *moveable*, but when in use the space behind the chair is critical (because the human sitting on it might move back).

$$\Psi_a(t_i|E) = \begin{cases} \Psi_{a,ang,h}\ \Psi_{a,ang,o}\ \Psi_{a,\beta} \\ \text{if } a \in \text{activities with human and} \\ \qquad \text{object at distance.} \\ \\ \Psi_{a,ang,h}\ \Psi_{a,dist,h} \\ \text{if } a \in \text{activities with human and} \\ \qquad \text{object in close proximity.} \end{cases} \tag{3.2}$$

We consider the planning affordance for several activities (e.g., *watching*, *interacting*, *working*, *sitting*, etc.). For each activity we model a separate cost function Ψ_a and evaluate trajectories using Eq. (3.1). We consider two classes of

activities: in the first, the human and object are in close proximity (*sitting, working, reaching etc.*) and in the second, they are at a distance (*walking, watching, interacting etc.*). The affordance varies with the distance and angle between the human and the object. We parameterize the cost as follows:

Angular preference $\Psi_{a,ang}(\cdot)$*:* Certain angular positions w.r.t. the human and the object are more relevant for certain activities. For example, the spatial distribution for the *watching* activity is spread over a wider angle than the *interacting* activity (see Figure 3.5). We capture the angular distribution of the activity in the space separating the human and the object with two cost functions $\Psi_{a,ang,h}$ and $\Psi_{a,ang,o}$, centered at human and object respectively. For activities with close-proximity between the human and the object we define a single cost centered at human. We parameterize the angular preference cost using the *von-Mises* distribution as:

$$\Psi_{a,ang,(\cdot)}(\mathbf{x}_{t_i}; \mu, \kappa) = \frac{1}{2\pi I_0(\kappa)} \exp(\kappa \mu^T \mathbf{x}_{t_i}) \tag{3.3}$$

In the above equation, μ and κ are parameters that we will learn from the data, and $\mathbf{x}_{t_i}$ is a two-dimensional unit vector. As illustrated in Figure 3.6, we obtain $\mathbf{x}_{t_i}$ by projecting the waypoint t_i onto the co-ordinate frame (x and y axis) defined locally for the human-object activity.

Distance preference $\Psi_{a,dist}$*:* The preferences vary with the robot distance from the human and the object. Humans do not prefer robots very close to them, especially when the robot is right-in-front or passes from behind [204]. Figure 3.5c shows the cost function learned by PlanIt. It illustrates that working humans do not prefer robots passing them from behind. We capture this by adding a 1D-Gaussian parameterized by a mean and variance, and centered at human.

Edge preference $\Psi_{a,\beta}$*:* For activities where the human and the object are separated

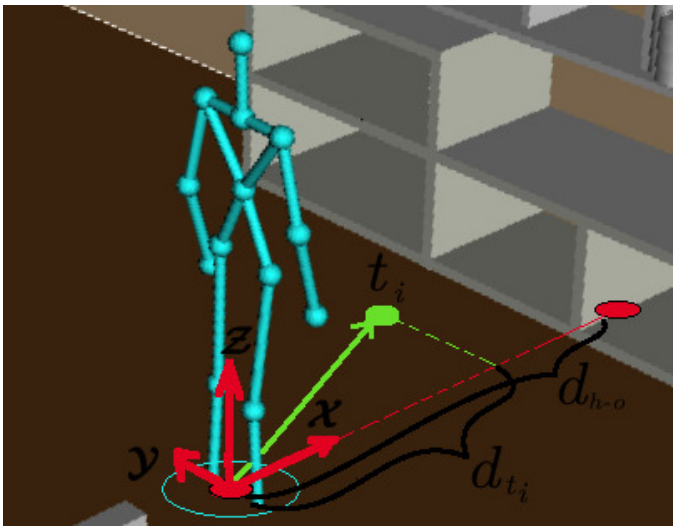
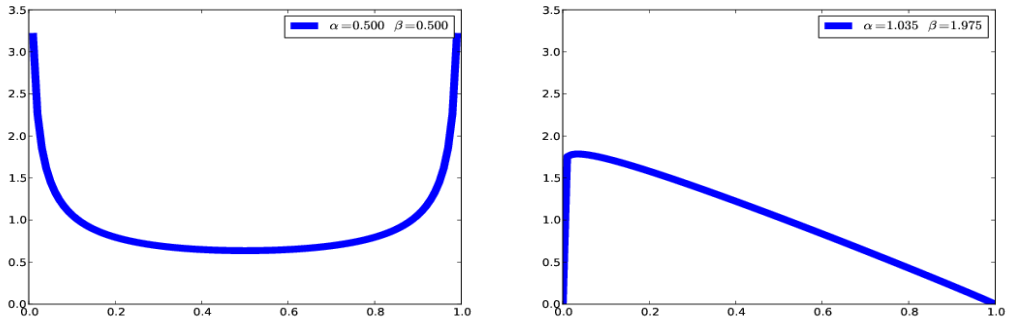


Figure 3.6: **Human side of local co-ordinate system for watching activity.** Similar co-ordinates are defined for the object human interacts with. Unit vector $\mathbf{x}_{t_i}$ is the projection of waypoint t_i on x-y axis and normalized it by its length. Distance between human and object is d_{h-o}, and t_i projected on x-axis is of length d_{t_i}.

by a distance, the preferences vary along the line connecting human and object. We parameterize this cost using a beta distribution which captures the relevance of the activity along the human-object edge. Figure 3.7 illustrates that in the *watching* activity users prefer robots to cross farther away from them, whereas for the *interacting* activity the preference is symmetric w.r.t. the humans. To calculate this cost for the waypoint t_i, we first take its distance from the human and project it along the line joining the human and the object d_{t_i}, and then normalize it by the distance d_{h-o} between the human and the object. The normalized distance is $\bar{d}_{t_i} = d_{t_i}/d_{h-o}$. In the equation below, we learn the parameters α and β.

$$\Psi_{\alpha,\beta}(\bar{d}_{t_i};\alpha,\beta) = \frac{\bar{d}_{t_i}^{\alpha-1}(1-\bar{d}_{t_i})^{\beta-1}}{\mathrm{B}(\alpha,\beta)}; \quad \bar{d}_{t_i} \in [0,1] \tag{3.4}$$

The functions used in Eq. (3.1) thus define our cost function. This, however, has many parameters (30) that need to be learned from data.



(a) Interacting (b) Watching

Figure 3.7: **Result of learned edge preference.** Distance between human and object is normalized to 1. Human is at 0 and object at 1. For interacting activity, edge preference is symmetric between two humans, but for watching activity humans do not prefer the robot passing very close to them.

3.5.2 Generative model

Given the user preference data from PlanIt we learn the parameters of Eq. (3.1). In order to keep the data collection easy we only elicit labels (bad, neutral or good) on the segments of the videos. The users do not reveal the human activity they think is being affected by the trajectory waypoint they labeled. In fact a waypoint can influence multiple activities. As illustrated in Figure 3.8 a waypoint between the humans and the TV can affect multiple watching activities.

We define a latent random variable $z_a^i \in \{0, 1\}$ for the waypoint t_i; which is 1 when the waypoint t_i affects the activity a and 0 otherwise. From the user preference data we learn the following cost function:

$$\mathbf{\Psi}(\{t_1, .., t_k\}|E) = \prod_{i=1}^{k} \underbrace{\sum_{a \in \mathcal{A}_E} p(z_a^i|E)\ \Psi_a(t_i|E)}_{\text{Marginalizing latent variable } z_a^i} \tag{3.5}$$

In the above equation, $p(z_a^i|E)$ (denoted with η_a) is the (prior) probability of user

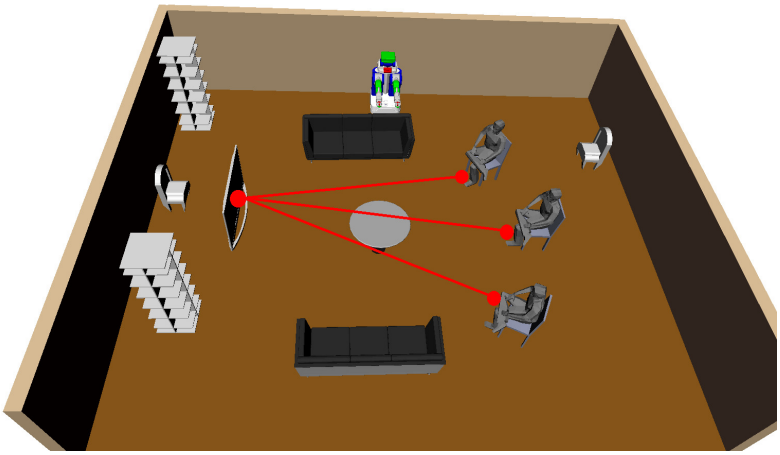


Figure 3.8: **Watching activity example.** Three humans watching a TV.

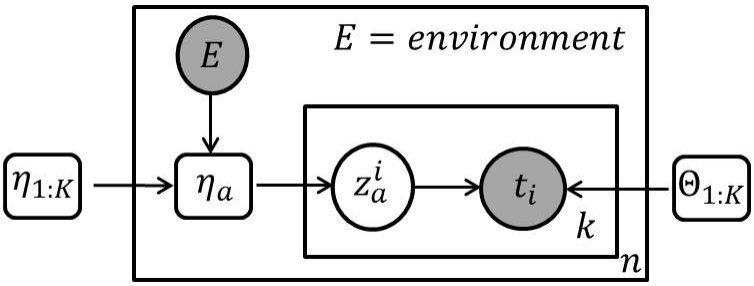


Figure 3.9: **Feedback model.** Generative model of the user preference data.

data arising from activity a, and $\mathcal{A}_E$ is the set of activities in environment E.[3] Figure 3.9 shows the generative process for preference data.

Training data: We obtain users preferences over n environments $E_1, .., E_n$. For each environment E we consider m trajectory segments $\mathcal{T}_{E,1}, .., \mathcal{T}_{E,m}$ labeled as bad by users. For each segment $\mathcal{T}$ we sample k waypoints $\{t_{\mathcal{T},1}, .., t_{\mathcal{T},k}\}$. We use $\Theta \in \mathbb{R}^{30}$ to denote the model parameters and solve the following maximum

[3]We extract the information about the environment and activities by querying OpenRAVE. In practice and in the robotic experiments, human activity information can be obtained using the software package by Koppula et al. [119].

likelihood problem:

$$\begin{aligned}\Theta^* &= \arg\max_{\Theta} \prod_{i=1}^{n} \prod_{j=1}^{m} \mathbf{\Psi}(\mathcal{T}_{E_i,j}|E_i;\Theta) \\ &= \arg\max_{\Theta} \prod_{i=1}^{n} \prod_{j=1}^{m} \prod_{l=1}^{k} \sum_{a\in\mathcal{A}_{E_i}} p(z_a^l|E_i;\Theta) \\ &\qquad \Psi_a(t_{\mathcal{T}_{E_i,j,l}}|E_i;\Theta)\end{aligned} \tag{3.6}$$

Eq. (3.6) does not have a closed form solution. We follow the Expectation-Maximization (EM) procedure to learn the model parameters. In the E-step we calculate the posterior activity assignment $p(z_a^l|t_{\mathcal{T}_{E_i,j,l}}, E_i)$ for all the waypoints, and in the M-step we update the parameters.

E-step: In this step, with fixed model parameters, we calculate the posterior probability of an activity being affected by a waypoint, as follows:

$$p(z_a|t, E;\Theta) = \frac{p(z_a|E;\Theta)\Psi_a(t|E;\Theta)}{\sum_{a\in\mathcal{A}_E} p(z_a|E;\Theta)\Psi_a(t|E;\Theta)} \tag{3.7}$$

We calculate the above probability for every activity a and the waypoint t labeled by users in our data set.

M-step: Using the probabilities calculated in the E-step we update the model parameters in the M-step. Our affordance representation consists of three distributions, namely: Gaussian, von-Mises and Beta. We update the parameters of the Gaussian, and the mean (μ) of the von-Mises in closed form. To update the variance (κ) of the von-Mises we follow the first order approximation proposed by Sra [206]. Finally the parameters of the beta distribution (α and β) are updated approximately by using the first and the second order moments of the data. As an example below we give the M-step update of the mean μ_a of the von-Mises.

$$\mu_a = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{l=1}^{k} p(z_a^l|\{t_{\mathcal{T}_{E_i,j,l}}\}, E_i)\mathbf{x}_{\{t_{\mathcal{T}_{E_i,j,l}}\}}}{\|\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{l=1}^{k} p(z_a^l|\{t_{\mathcal{T}_{E_i,j,l}}\}, E_i)\mathbf{x}_{\{t_{\mathcal{T}_{E_i,j,l}}\}}\|} \tag{3.8}$$

3.6 Experiments on PlanIt

Our data set consists of 112 context-rich 3D-environments that resemble real living rooms or bedrooms. We create them by downloading 2D-images of real environments from the Google images and reconstructing their corresponding 3D models using OpenRAVE [48].[4] Figure 3.10 shows some example environments and their corresponding 2D images. We depict human activities by adding to the 3D models different human poses obtained from the Kinect (refer Figure 3 in [100] for the human poses). In our experiments we consider six activities: *walking*, *watching*, *interacting*, *reaching*, *sitting* and *working* as shown in Figure 3.1.

For these environments we generate trajectory videos and add them to the PlanIt database. We crowdsource 2500 trajectory videos through PlanIt and for each trajectory a user labeled segments of it as *bad*, *neutral* or *good*, corresponding to the scores 1, 3 and 5 respectively.

3.6.1 Baseline algorithms

We consider the following baseline cost functions:

- *Chance*: Uniformly randomly assigns a cost in interval [0,1] to a trajectory.
- *Maximum Clearance Planning (MCP)*: Inspired by Sisbot et al. [204], this heuristic favors trajectories which stay farther away from objects. The MCP cost of a trajectory is the (negated) root mean square distance from the nearest object across the trajectory waypoints.

[4]For reconstructing 3D environments we download 3D object (.obj) files off the web, mainly from the Google warehouse.

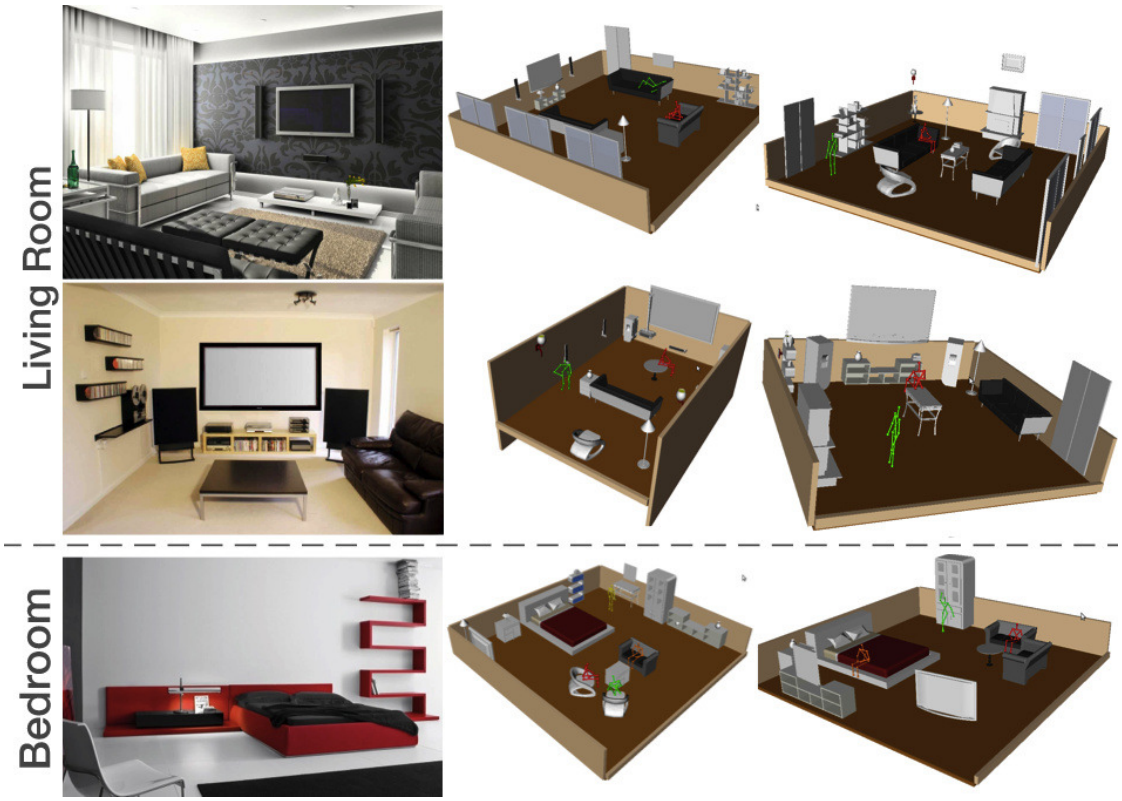


Figure 3.10: **Examples from our dataset:** four living room and two bedroom environments. On left is the 2D image we download from Google images. On right are the 3D reconstructed environments in OpenRAVE. All environments are rich in the types and number of objects and often have multiple humans perform different activities.

- *Human Interference Count (HIC)*: HIC cost of a trajectory is the number of times it interferes with human activities. Interfering rules were hand designed on expert's opinion.
- *Metropolis Criterion Costmap (MCC):* Similar to Mainprice et al. [150], a trajectory's cost exponentially increases with its closeness to surrounding objects. The MCC cost of a trajectory is defined as follows: c_{t_i} =

$\min_{o \in O} dist(t_i, o)$

$$\Psi_{mc}(t_i) = \begin{cases} e^{-c_{t_i}} & c_{t_i} < 1m \\ 0 & \text{otherwise} \end{cases}$$

$$MCC(\mathcal{T} = \{t_1, .., t_n\}) = \frac{\sum_{i=1}^{n} \Psi_{mc}(t_i)}{n}$$

$dist(t_i, o)$ is the euclidean distance between the waypoint t_i and the object o.

- *HIC scaled with MCC:* We design this heuristic by combining the HIC and the MCC costs. The HICMCC cost of a trajectory is $HICMCC(\mathcal{T}) = MCC(\mathcal{T}) * HIC(\mathcal{T})$
- *Trajectory Preference Perceptron (TPP):* Jain et al. [96] learns a cost function from co-active user feedback in an online setting. We compare against the TPP using trajectory features from [96].

The above described baselines assign cost to trajectories and lower cost is preferred. For quantitative evaluation each trajectory is also assigned a ground truth score based on the user feedback from PlanIt. The ground truth score of a trajectory is the minimum score given by a user.[5] For example if two segments of a trajectory are labeled with scores 3 (neutral) and 5 (good), then the ground truth score is 3. We denote the ground truth score of trajectory $\mathcal{T}$ as $score(\mathcal{T})$.

3.6.2 Evaluation metric

Given the ground truth scores we evaluate algorithms based on the following metrics.

[5]The rationale behind this definition of the ground truth score is that a trajectory with a single bad waypoint is considered to be overall bad.

- *Misclassification rate:* For a trajectory $\mathcal{T}_i$ we consider the set of trajectories $\mathbf{T}_i$ with higher ground truth score: $\mathbf{T}_i = \{\mathcal{T} | score(T) > score(\mathcal{T}_i)\}$. The misclassification rate of an algorithm is the number of trajectories in $\mathbf{T}_i$ which it assigns a higher cost than $\mathcal{T}_i$. We normalize this count by the number of trajectories in $\mathbf{T}_i$ and average it over all the trajectories $\mathcal{T}_i$ in the data set. Lower misclassification rate is desirable.
- *Normalized discounted cumulative gain (nDCG) [152]:* This metric quantifies how well an algorithm rank trajectories. It is a relevant metric because autonomous robots can rank trajectories and execute the top ranked trajectory [47, 96]. Obtaining a rank list simply amounts to sorting the trajectories based on their costs.

3.6.3 Results

We evaluate the trained model for its:

- *Discriminative power*: How well can the model distinguish good/bad trajectories?
- *Interpretability*: How well does the qualitative visualization of the cost function heatmaps match our intuition?

3.6.4 Discriminative power of learned cost function

A model trained on users preferences should reliably distinguish good from bad trajectories i.e. if we evaluate two trajectories under the learned model then it

Table 3.1: **Misclassification rate**: chances that an algorithm presented with two trajectories (one good and other bad) orders them incorrectly. Lower rate is better. The number inside bracket is standard error.

Algorithms	Bedroom	Living room
Chance	.52 (-)	.48 (-)
MCP based on Sisbot et al. [204]	.46 (.04)	.42 (.06)
MCC based on Mainprice et al. [150]	.44 (.03)	.42 (.06)
HIC	.30 (.04)	.23 (.06)
HICMCC	.32 (.04)	.29 (.05)
TPP based on Jain et al. [96]	.33 (.03)	.34 (.05)
Ours within scenario evaluation	.32 (.05)	.19 (.03)
Ours cross scenario evaluation	**.27 (.04)**	**.17 (.05)**

should assign a lower cost to the better trajectory. We compare algorithms under two training settings: (i) *within-env*: we test and train on the same category of environment using 5-fold cross validation, e.g. training on bedrooms and testing on new bedrooms; and (ii) *cross-env*: we train on bedrooms and test on living rooms, and vice-versa. In both settings the algorithms were tested on environments not seen before.

How well algorithms discriminate between trajectories? We compare algorithms on the evaluation metrics described above. As shown in Table 3.1 our algorithm gives the lowest misclassification rate in comparison to the baseline algorithms. We also observe that the misclassification rate on bedrooms is lower with cross-env training than within-env. We conjecture this is because of a harder learning problem (on average) when training on bedrooms than living rooms. In our data set, on average a bedroom have 3 human activities while a living room have 2.06. Therefore the model parameters converge to better optima when trained on living rooms. As illustrated in Figure 3.11, our algorithm

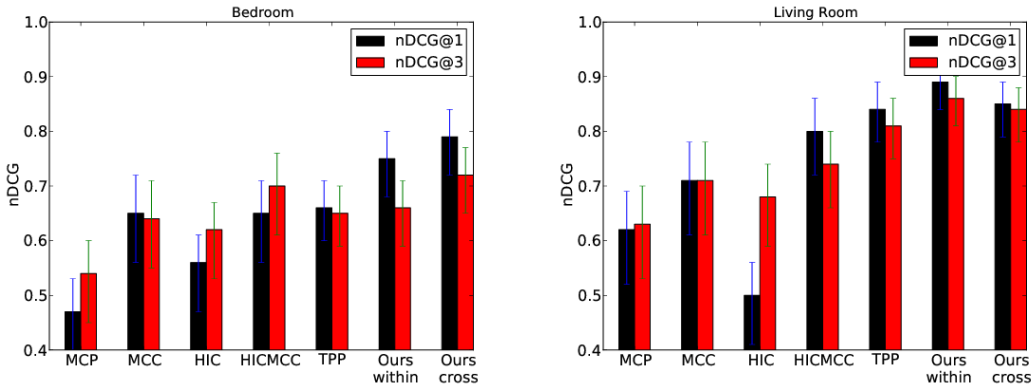


Figure 3.11: **nDCG plots** comparing algorithms on bedroom (left) and living room (right) environments. Error bar indicates standard error.

also ranks trajectories better than other baseline algorithms.

Crowdsourcing helps and so does learning preferences! We compare our approach to TPP learning algorithm by Jain et al. [96]. TPP learns with co-active feedback which requires the user to iteratively improve the trajectory proposed by the system. This feedback is time consuming to elicit and therefore difficult to scale to many environments. On both evaluation metrics our crowdsourcing approach outperforms TPP, which is trained on fewer environments. Figure 3.12 shows our model improves as users provide more feedback through PlanIt. Our data-driven approach is also a better alternative to hand-designed cost functions.

3.6.5 Interpretability: Qualitative visualization of learned cost

Visualizing the learned heatmaps is useful for understanding the spatial distribution of human-object activities. We discussed the learned heatmaps for watching, interacting and working activities in Section 3.5.1 (see Figure 3.5).

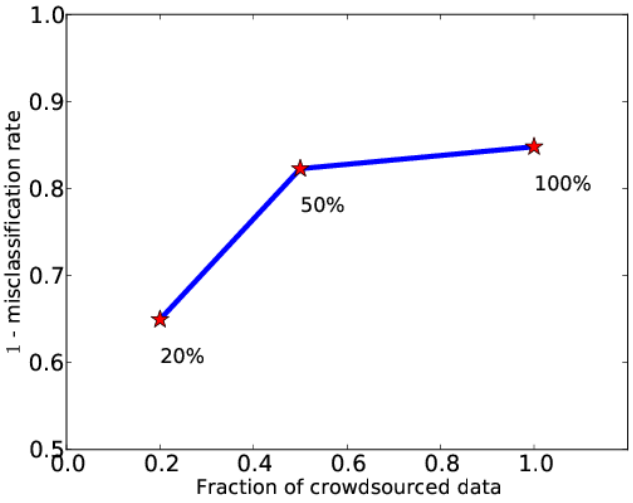
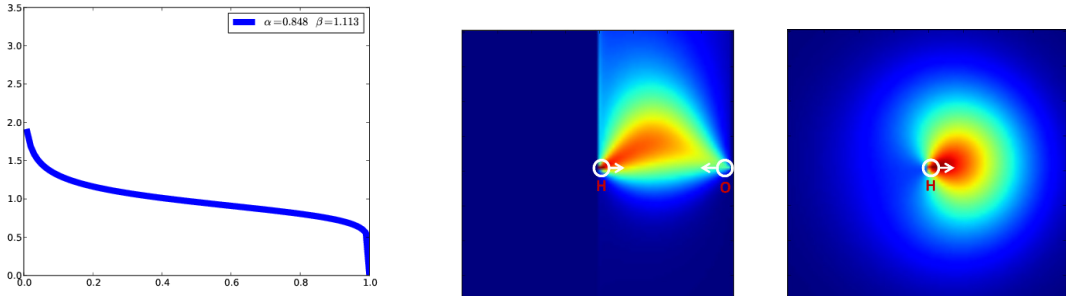


Figure 3.12: **Crowdsourcing improves performance**: Misclassification rate decreases as more users provide feedback via PlanIt.

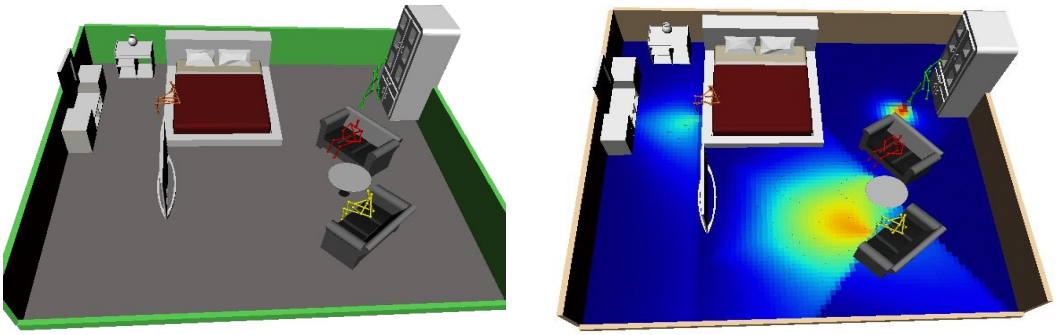


(a) Edge preference for walking (b) Walking (c) Sitting

Figure 3.13: **Learned affordance heatmaps**. (a) Edge preference for walking activity. Human is at 0 and object at 1. (b,c) Top view of heatmaps. Human is at the center and facing right.

Figure 3.13 illustrates the heatmap for the walking, and sitting activities.

How does crowd preferences change with the nature of activity? For the same distance between the human and the object, the spatial preference learned from the crowd is less-spread for interacting activity than watching and walking activities, as shown in Figure 3.5. This empirical evidence implies that while



(a) Bedroom (b) Planning affordance

Figure 3.14: **Planning affordance map for an environment**. Bedroom with sitting, reaching and watching activities. A robot uses the affordance map as a cost for planning good trajectories.

interacting humans do not mind the robot in vicinity unless it blocks their eye contact. The preferences also vary along the line joining the human and object. As shown in Figure 3.7, while watching TV the space right in front of the human is critical, and the edge-preference rapidly decays as we move towards the TV. On the other hand, when human is walking towards an object the decay in edge-preference is slower, Figure 3.13(a).

Using the PlanIt system and the cost map of individual activities, a.k.a. affordance library, we generate the planning map of environments with multiple activities. Some examples of the planning map are shown in Figure 3.14.

3.6.6 Robotic experiment

In order to plan trajectories in an unseen environment we generate its planning map and use it as an input cost to the RRT [137] planner. Given the cost function, RRT plans trajectories with low cost. We implement our learned model on PR2 robot for the purpose of navigation when the human is watching a football

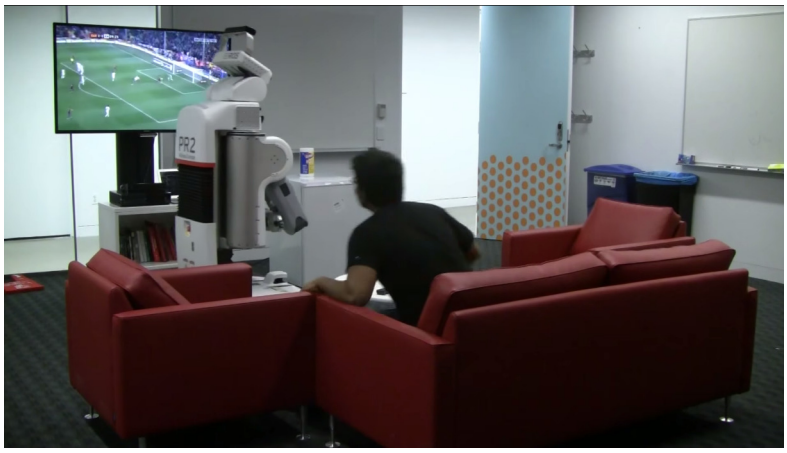


Figure 3.15: **Robotic experiment:** A screen-shot of our algorithm running on PR2. Without learning robot blocks the view of the human watching football. `http://planit.cs.cornell.edu/video`

match on the TV (Figure 3.15). Without learning the preferences, the robot plans the shortest path to the goal and obstructs the human's view of TV. Demonstration: `http://planit.cs.cornell.edu/video`

3.7 Conclusion

In this chapter we proposed a crowdsourcing approach for learning user preferences over trajectories. Our PlanIt system is user-friendly and easy-to-use for eliciting large scale preference data from non-expert users. The simplicity of PlanIt comes at the expense of weak and noisy labels with latent annotator intentions. We presented a generative approach with latent nodes to model the preference data. Through PlanIt we learn spatial distribution of activities involving humans and objects. We experimented on many context-rich living and bedroom environments. Our results validate the advantages of crowdsourcing and learning the cost over hand encoded heuristics. We also implemented

our learned model on the PR2 robot. PlanIt is publicly available for visualizing learned cost function heatmaps, viewing robotic demonstration, and providing preference feedback `http://planit.cs.cornell.edu`

CHAPTER 4

ANTICIPATING MANEUVERS FROM IMPLICIT DRIVING SIGNALS

Driving a car is one of the most naturally occurring interaction between a human and a robot (i.e. the car). In this chapter we address the question: *what can a car learn by observing its driver?*

Most of the modern cars comes equipped with sensors like radars, cameras, GPS etc., that can automatically detect maneuvers like lane changes, turns, braking etc. By observing many drivers drive their cars, we learn to anticipate driving maneuvers before they happen. It turns out that eliciting training examples from the drivers for this task is a very natural process. We let drivers drive the car to their destination and they implicitly provide many examples of maneuvers (lane changes, turns etc.).

4.1 Motivation for maneuver anticipation

Advanced Driver Assistance Systems (ADAS) have made driving safer over the last decade. They prepare vehicles for unsafe road conditions and alert drivers if they perform a dangerous maneuver. However, many accidents are unavoidable because by the time drivers are alerted, it is already too late. Anticipating maneuvers beforehand can alert drivers before they perform the maneuver and also give ADAS more time to avoid or prepare for the danger.

In this chapter we propose a vehicular sensor-rich platform and learning algorithms for maneuver anticipation. For this purpose we equip a car with cameras, Global Positioning System (GPS), and a computing device to capture the driving context from both inside and outside of the car. In order to antici-

pate maneuvers, we propose a sensory-fusion deep learning architecture which jointly learns to anticipate and fuse multiple sensory streams. Our architecture consists of Recurrent Neural Networks (RNNs) that use Long Short-Term Memory (LSTM) units to capture long temporal dependencies. We propose a novel training procedure which allows the network to predict the future given only a partial temporal context. We introduce a diverse data set with 1180 miles of natural freeway and city driving, and show that we can anticipate maneuvers 3.5 seconds before they occur in real-time with a precision and recall of 90.5% and 87.4% respectively.

4.2 Robotic anticipation

Over the last decade cars have been equipped with various assistive technologies in order to provide a safe driving experience. Technologies such as lane keeping, blind spot check, pre-crash systems etc., are successful in alerting drivers whenever they commit a dangerous maneuver [136]. Still in the US alone more than 33,000 people die in road accidents every year, the majority of which are due to inappropriate maneuvers [167]. We therefore need mechanisms that can alert drivers *before* they perform a dangerous maneuver in order to avert many such accidents [190].

In this chapter we address the problem of anticipating maneuvers that a driver is likely to perform in the next few seconds. Figure 4.1 shows our system anticipating a left turn maneuver a few seconds before the car reaches the intersection. Our system also outputs probabilities over the maneuvers the driver can perform. With this prior knowledge of maneuvers, the driver assistance sys-

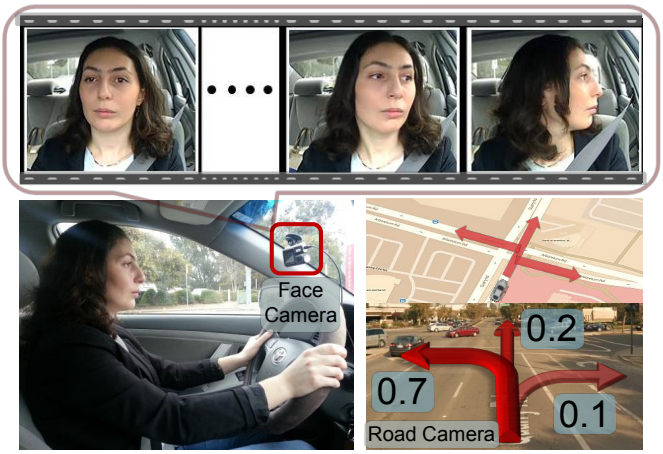


Figure 4.1: **Anticipating maneuvers.** Our algorithm anticipates driving maneuvers performed a few seconds in the future. It uses information from multiple sources including videos, vehicle dynamics, GPS, and street maps to anticipate the probability of different future maneuvers.

tems can alert drivers about possible dangers before they perform the maneuver, thereby giving them more time to react. Some previous works [68, 132, 163] also predict a driver's future maneuver. However, as we show in the following sections, these methods use limited context and/or do not accurately model the anticipation problem.

In order to anticipate maneuvers, we reason with the contextual information from the surrounding events, which we refer to as the *driving context*. We obtain this driving context from multiple sources. We use videos of the driver inside the car and the road in front, the vehicle's dynamics, global position coordinates (GPS), and street maps; from this we extract a time series of multi-modal data from both inside and outside the vehicle. The challenge lies in modeling the temporal aspects of driving and fusing the multiple sensory streams. In this work we propose a specially tailored approach for anticipation in such sensory-rich settings.

Anticipation of the future actions of a human is an important perception task with applications in robotics and computer vision [130, 255, 113, 121, 244]. It requires the prediction of future events from a limited temporal context. This differentiates anticipation from *activity recognition* [244], where the complete temporal context is available for prediction. Furthermore, in sensory-rich robotics settings like ours, the context for anticipation comes from multiple sensors. In such scenarios the end performance of the application largely depends on how the information from different sensors are fused. Previous works on anticipation [113, 121, 130] usually deal with single-data modality and do not address anticipation for sensory-rich robotics applications. Additionally, they learn representations using shallow architectures [91, 113, 121, 130] that cannot handle long temporal dependencies [13].

In order to address the anticipation problem more generally, we propose a Recurrent Neural Network (RNN) based architecture which learns rich representations for anticipation. We focus on sensory-rich robotics applications, and our architecture learns how to optimally fuse information from different sensors. Our approach captures temporal dependencies by using Long Short-Term Memory (LSTM) units. We train our architecture in a sequence-to-sequence prediction manner (Figure 4.2) such that it explicitly learns to anticipate given a partial context, and we introduce a novel loss layer which helps anticipation by preventing over-fitting.

We evaluate our approach on a driving data set with 1180 miles of natural freeway and city driving collected across two states – from 10 drivers and with different kinds of driving maneuvers. The data set is challenging because of the variations in routes and traffic conditions, and the driving styles of the drivers

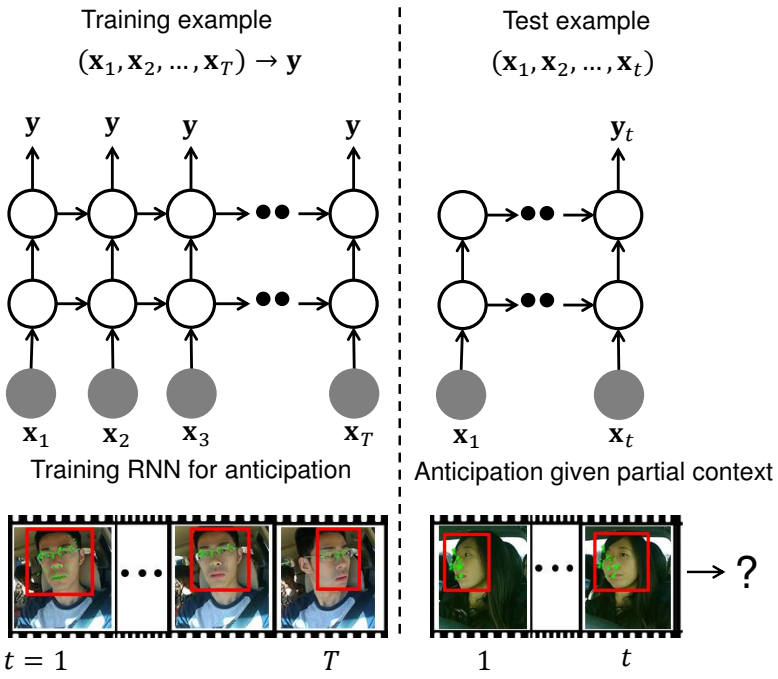


Figure 4.2: (**Left**) Shows training RNN for anticipation in a sequence-to-sequence prediction manner. The network explicitly learns to map the partial context $(\mathbf{x}_1, .., \mathbf{x}_t)$ $\forall t$ to the future event $\mathbf{y}$. (**Right**) At test time the network's goal is to anticipate the future event as soon as possible, i.e. by observing only a partial temporal context.

(Figure 4.3). We demonstrate that our deep learning sensory-fusion approach anticipates maneuvers 3.5 seconds before they occur with 84.5% precision and 77.1% recall while using out-of-the-box face tracker. With more sophesticated 3D pose estimation of the face, our precision and recall increases to **90.5%** and **87.4%** respectively. We believe that our work creates scope for new ADAS features to make roads safer. In summary our key contributions are as follows:

- We propose an approach for anticipating driving maneuvers several seconds in advance.
- We propose a generic sensory-fusion RNN-LSTM architecture for anticipation in robotics applications.

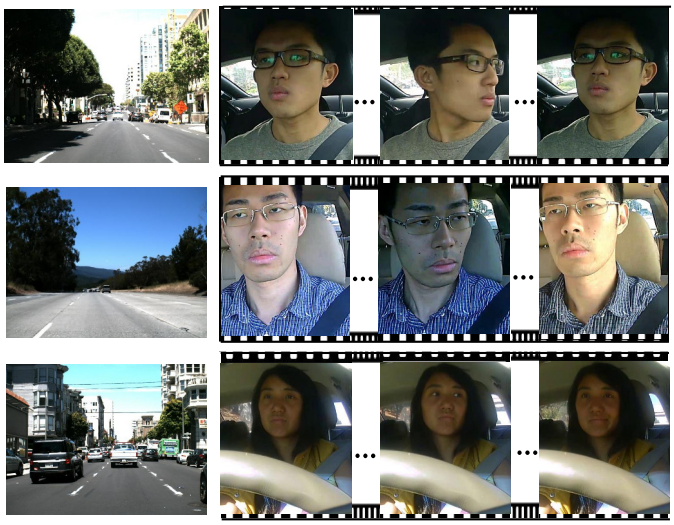


Figure 4.3: **Variations in the data set.** Images from the data set [91] for a left lane change. (**Left**) Views from the road facing camera. (**Right**) Driving style of the drivers vary for the same maneuver.

- We release the first data set of natural driving with videos from both inside and outside the car, GPS, and speed information.
- We release an open-source deep learning package `NeuralModels` which is especially designed for robotics applications with multiple sensory streams.

Our data set and deep learning code are publicly available at: `http://www.brain4cars.com`

4.3 Assistive cars and related works

Our work builds upon the previous works on assisitive vehicular technologies, anticipating human activities, learning temporal models, and computer vision methods for analyzing human face.

Assistive features for vehicles. Latest cars available in market comes equipped with cameras and sensors to monitor the surrounding environment. Through multi-sensory fusion they provide assisitive features like lane keeping, forward collision avoidance, adaptive cruise control etc. These systems warn drivers when they perform a potentially dangerous maneuver [198, 235]. Driver monitoring for distraction and drowsiness has also been extensively researched [66, 185]. Techniques like eye-gaze tracking are now commercially available (Seeing Machines Ltd.) and has been effective in detecting distraction. Our work complements existing ADAS and driver monitoring techniques by anticipating maneuvers several seconds before they occur.

Closely related to us are previous works on predicting the driver's intent. Vehicle trajectory has been used to predict the intent for lane change or turn maneuver [20, 68, 132, 145]. Most of these works ignore the rich context available from cameras, GPS, and street maps. Previous works have addressed maneuver anticipation [1, 163, 51, 230] through sensory-fusion from multiple cameras, GPS, and vehicle dynamics. In particular, Morris et al. [163] and Trivedi et al. [230] used Relevance Vector Machine (RVM) for intent prediction and performed sensory fusion by concatenating feature vectors. We will show that such hand designed concatenation of features does not work well. Furthermore, these works do not model the temporal aspect of the problem properly. They assume that informative contextual cues always appear at a fixed time before the maneuver. We show that this assumption is not true, and in fact the temporal aspect of the problem should be carefully modeled. In contrast to these works, our RNN-LSTM based sensory-fusion architecture captures long temporal dependencies through its memory cell and learns rich representations for anticipation through a hierarchy of non-linear transformations of input data.

Our work is also related to works on driver behavior prediction with different sensors [87, 66, 65], and vehicular controllers which act on these predictions [198, 235, 56].

Anticipation and Modeling Humans. Modeling of human motion has given rise to many applications, anticipation being one of them. Anticipating human activities has shown to improve human-robot collaboration [244, 123, 149, 120, 54]. Similarly, forecasting human navigation trajectories has enabled robots to plan sociable trajectories around humans [113, 17, 130, 90]. Feature matching techniques have been proposed for anticipating human activities from videos [192]. Modeling human preferences has enabled robots to plan good trajectories [55, 204, 94, 93]. Similar to these works, we anticipate human actions, which are driving maneuvers in our case. However, the algorithms proposed in the previous works do not apply in our setting. In our case, anticipating maneuvers requires modeling the interaction between the driving context and the driver's intention. Such interactions are absent in the previous works, and they use shallow architectures [13] that do not properly model temporal aspects of human activities. They further deal with a single data modality and do not tackle the challenges of sensory-fusion. Our problem setup involves all these challenges, for which we propose a deep learning approach which efficiently handles temporal dependencies and learns to fuse multiple sensory streams.

Analyzing the human face. The vision approaches related to our work are face detection and tracking [238, 250], statistical models of face [38] and pose estimation methods for face [248]. Active Appearance Model (AAM) [38] and its variants [153, 247] statistically model the shape and texture of the face. AAMs have also been used to estimate the 3D-pose of a face from a single image [248]

and in design of assistive features for driver monitoring [185, 218]. In our approach we adapt off-the-shelf available face detection [238] and tracking algorithms [196] (see Section 4.7). Our approach allows us to easily experiment with more advanced face detection and tracking algorithms. We demonstrate this by using the Constrained Local Neural Field (CLNF) model [9] and tracking 68 fixed landmark points on the driver's face and estimating the 3D head-pose.

Learning temporal models. Temporal models are commonly used to model human activities [122, 162, 242, 243]. These models have been used in both discriminative and generative fashions. The discriminative temporal models are mostly inspired by the Conditional Random Field (CRF) [133] which captures the temporal structure of the problem. Wang et al. [243] and Morency et al. [162] propose dynamic extensions of the CRF for image segmentation and gesture recognition respectively. On the other hand, generative approaches for temporal modeling include various filtering methods, such as Kalman and particle filters [225], Hidden Markov Models, and many types of Dynamic Bayesian Networks [164]. Some previous works [20, 131, 171] used HMMs to model different aspects of the driver's behaviour. Most of these generative approaches model how latent (hidden) states influence the observations. However, in our problem both the latent states and the observations influence each other. In the following sections, we will describe the Autoregressive Input-Output HMM (AIO-HMM) for maneuver anticipation [91] and will use it as a baseline to compare our deep learning approach. Unlike AIO-HMM our deep architecture have internal memory which allows it to handle long temporal dependencies [83]. Furthermore, the input features undergo a hierarchy of non-linear transformation through the deep architecture which allows learning rich representations.

Two building blocks of our architecture are Recurrent Neural Networks (RNNs) [174] and Long Short-Term Memory (LSTM) units [84]. Our work draws upon ideas from previous works on RNNs and LSTM from the language [214], speech [81], and vision [49] communities. Our approach to the joint training of multiple RNNs is related to the recent work on hierarchical RNNs [57]. We consider RNNs in multi-modal setting, which is related to the recent use of RNNs in image-captioning [49]. Our contribution lies in formulating activity anticipation in a deep learning framework using RNNs with LSTM units. We focus on sensory-rich robotics applications, and our architecture extends previous works doing sensory-fusion with feed-forward networks [166, 212] to the fusion of temporal streams. Using our architecture we demonstrate state-of-the-art on maneuver anticipation.

4.4 Maneuver anticipation

We first give an overview of the maneuver anticipation problem and then describe our system.

4.4.1 Problem overview

Our goal is to anticipate driving maneuvers a few seconds before they occur. This includes anticipating a lane change before the wheels touch the lane markings or anticipating if the driver keeps straight or makes a turn when approaching an intersection. This is a challenging problem for multiple reasons. First, it requires the modeling of context from different sources. Information from a sin-

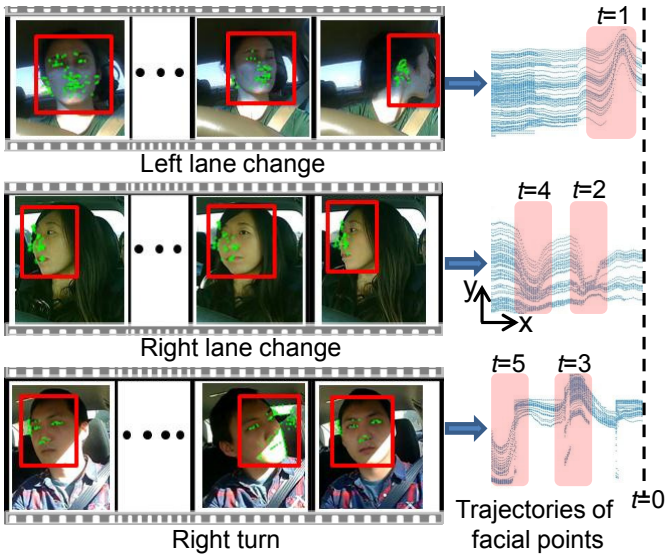


Figure 4.4: **Variable time occurrence of events.** *Left*: The events inside the vehicle before the maneuvers. We track the driver's face along with many facial points. *Right*: The trajectories generated by the horizontal motion of facial points (pixels) 't' seconds before the maneuver. X-axis is the time and Y-axis is the pixels' horizontal coordinates. Informative cues appear during the shaded time interval. Such cues occur at variable times before the maneuver, and the order in which the cues appear is also important.

gle source, such as a camera capturing events outside the car, is not sufficiently rich. Additional visual information from within the car can also be used. For example, the driver's head movements are useful for anticipation – drivers typically check for the side traffic while changing lanes and scan the cross traffic at intersections.

Second, reasoning about maneuvers should take into account the driving context at both local and global levels. Local context requires modeling events in vehicle's vicinity such as the surrounding vision, GPS, and speed information. On the other hand, factors that influence the overall route contributes to the global context, such as the driver's final destination. Third, the informative

cues necessary for anticipation appear at variable times before the maneuver, as illustrated in Figure 4.4. In particular, the time interval between the driver's head movement and the occurrence of the maneuver depends on many factors such as the speed, traffic conditions, etc.

In addition, appropriately fusing the information from multiple sensors is crucial for anticipation. Simple sensory fusion approaches like concatenation of feature vectors performs poorly, as we demonstrate through experiments. In our proposed approach we learn a neural network layer for fusing the temporal streams of data coming from different sensors. Our resulting architecture is end-to-end trainable via back propagation, and we jointly train it to: (i) model the temporal aspects of the problem; (ii) fuse multiple sensory streams; and (iii) anticipate maneuvers.

4.4.2 System overview

For maneuver anticipation our vehicular sensory platform includes the following (as shown in Figure 4.5):

1. A driver-facing camera inside the vehicle. We mount this camera on the dashboard and use it to track the driver's head movements. This camera operates at 25 fps.
2. A camera facing the road is mounted on the dashboard to capture the (outside) view in front of the car. This camera operates at 30 fps. The video from this camera enables additional reasoning on maneuvers. For example, when the vehicle is in the left-most lane, the only safe maneuvers are

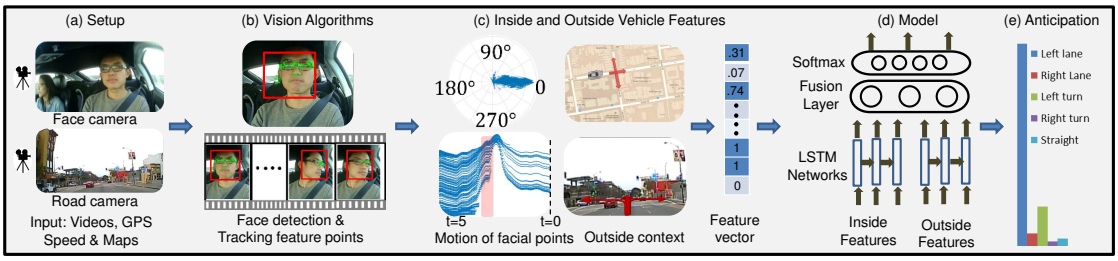


Figure 4.5: **System Overview.** Our system anticipating a left lane change maneuver. (a) We process multi-modal data including GPS, speed, street maps, and events inside and outside of the vehicle using video cameras. (b) Vision pipeline extracts visual cues such as driver's head movements. (c) The inside and outside driving context is processed to extract expressive features. (d,e) Using our deep learning architecture we fuse the information from outside and inside the vehicle and anticipate the probability of each maneuver.

a right-lane change or keeping straight, unless the vehicle is approaching an intersection.

3. A speed logger for vehicle dynamics because maneuvers correlate with the vehicle's speed, e.g., turns usually happen at lower speeds than lane changes.

4. A Global Positioning System (GPS) for localizing the vehicle on the map. This enables us to detect upcoming road artifacts such as intersections, highway exits, etc.

Using this system we collect 1180 miles of natural city and freeway driving data from 10 drivers. We denote the information from sensors with feature vector $\mathbf{x}$. Our vehicular systems gives a temporal sequence of feature vectors $\{(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t, ...)\}$. For now we do not distinguish between the information from different sensors, later in Section 4.6.2 we introduce sensory fusion. In Section 4.7 we formally define our feature representations and describe our data

set in Section 4.9.1. We now formally define anticipation and present our deep learning architecture.

4.5 Preliminaries

We now formally define anticipation and then present our Recurrent Neural Network architecture. The goal of anticipation is to predict an event several seconds before it happens given the contextual information up to the present time. The future event can be one of multiple possibilities. At training time a set of temporal sequences of observations and events $\{(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)_j, \mathbf{y}_j\}_{j=1}^{N}$ is provided where $\mathbf{x}_t$ is the observation at time t, $\mathbf{y}$ is the representation of the event (described below) that happens at the end of the sequence at $t = T$, and j is the sequence index. At test time, however, the algorithm receives an observation $\mathbf{x}_t$ at each time step, and its goal is to predict the future event as early as possible, i.e. by observing only a partial sequence of observations $\{(\mathbf{x}_1, ..., \mathbf{x}_t)|t < T\}$. This differentiates anticipation from *activity recognition* [240, 119] where in the latter the complete observation sequence is available at test time. In this chapter, $\mathbf{x}_t$ is a real-valued feature vector and $\mathbf{y} = [y^1, ..., y^K]$ is a vector of size K (the number of events), where y^k denotes the probability of the temporal sequence belonging to event the k such that $\sum_{k=1}^{K} y^k = 1$. At the time of training, $\mathbf{y}$ takes the form of a one-hot vector with the entry in $\mathbf{y}$ corresponding to the ground truth event as 1 and the rest 0.

In this chapter we propose a deep RNN architecture with Long Short-Term Memory (LSTM) units [84] for anticipation. Below we give an overview of the standard RNN and LSTM which form the building blocks of our architecture.

4.5.1 Recurrent Neural Networks

A standard RNN [174] takes in a temporal sequence of vectors $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)$ as input, and outputs a sequence of vectors $(\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_T)$ also known as high-level representations. The representations are generated by non-linear transformation of the input sequence from $t = 1$ to T, as described in the equations below.

$$\mathbf{h}_t = f(\mathbf{W}\mathbf{x}_t + \mathbf{H}\mathbf{h}_{t-1} + \mathbf{b}) \tag{4.1}$$

$$\mathbf{y}_t = \texttt{softmax}(\mathbf{W}_y\mathbf{h}_t + \mathbf{b}_y) \tag{4.2}$$

where f is a non-linear function applied element-wise, and $\mathbf{y}_t$ is the `softmax` probabilities of the events having seen the observations up to $\mathbf{x}_t$. $\mathbf{W}$, $\mathbf{H}$, $\mathbf{b}$, $\mathbf{W}_y$, $\mathbf{b}_y$ are the parameters that are learned. Matrices are denoted with bold, capital letters, and vectors are denoted with bold, lower-case letters. In a standard RNN a common choice for f is `tanh` or `sigmoid`. RNNs with this choice of f suffer from a well-studied problem of *vanishing gradients* [174], and hence are poor at capturing long temporal dependencies which are essential for anticipation. A common remedy to vanishing gradients is to replace `tanh` non-linearities by Long Short-Term Memory cells [84]. We now give an overview of LSTM and then describe our model for anticipation.

4.5.2 Long Short-Term Memory Cells

LSTM is a network of neurons that implements a memory cell [84]. The central idea behind LSTM is that the memory cell can maintain its state over time. When combined with RNN, LSTM units allow the recurrent network to remember long term context dependencies.

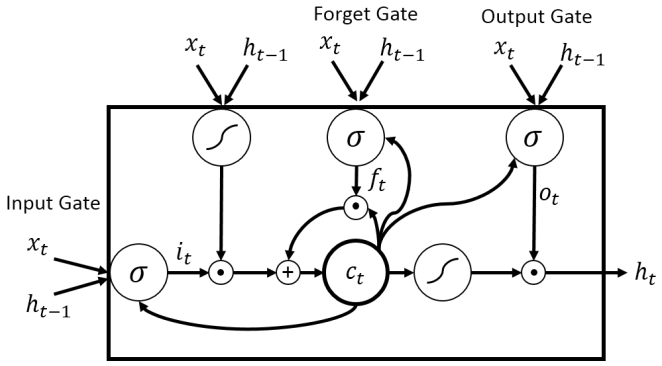


Figure 4.6: **Internal working of an LSTM unit.**

LSTM consists of three gates – input gate **i**, output gate **o**, and forget gate **f** – and a memory cell **c**. See Figure 4.6 for an illustration. At each time step t, LSTM first computes its gates' activations $\{\mathbf{i}_t,\mathbf{f}_t\}$ (4.3)(4.4) and updates its memory cell from $\mathbf{c}_{t-1}$ to $\mathbf{c}_t$ (4.5), it then computes the output gate activation $\mathbf{o}_t$ (4.6), and finally outputs a hidden representation $\mathbf{h}_t$ (4.7). The inputs into LSTM are the observations $\mathbf{x}_t$ and the hidden representation from the previous time step $\mathbf{h}_{t-1}$. LSTM applies the following set of update operations:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i\mathbf{x}_t + \mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{V}_i\mathbf{c}_{t-1} + \mathbf{b}_i) \tag{4.3}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{V}_f\mathbf{c}_{t-1} + \mathbf{b}_f) \tag{4.4}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \texttt{tanh}(\mathbf{W}_c\mathbf{x}_t + \mathbf{U}_c\mathbf{h}_{t-1} + \mathbf{b}_c) \tag{4.5}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{V}_o\mathbf{c}_t + \mathbf{b}_o) \tag{4.6}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \texttt{tanh}(\mathbf{c}_t) \tag{4.7}$$

where $\odot$ is an element-wise product and σ is the logistic function. σ and tanh are applied element-wise. $\mathbf{W}_*$, $\mathbf{V}_*$, $\mathbf{U}_*$, and $\mathbf{b}_*$ are the parameters, further the weight matrices $\mathbf{V}_*$ are diagonal. The input and forget gates of LSTM participate in updating the memory cell (4.5). More specifically, forget gate controls the part of memory to forget, and the input gate computes new values based

on the current observation that are written to the memory cell. The output gate together with the memory cell computes the hidden representation (4.7). Since LSTM cell activation involves *summation* over time (4.5) and derivatives distribute over sums, the gradient in LSTM gets propagated over a longer time before vanishing. In the standard RNN, we replace the non-linear f in equation (4.1) by the LSTM equations given above in order to capture long temporal dependencies. We use the following shorthand notation to denote the recurrent LSTM operation.

$$(\mathbf{h}_t, \mathbf{c}_t) = \text{LSTM}(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}) \tag{4.8}$$

We now describe our RNN architecture with LSTM units for anticipation. Following which we will describe a particular instantiation of our architecture for maneuver anticipation where the observations $\mathbf{x}$ come from multiple sources.

4.6 Network architecture for anticipation

In order to anticipate, an algorithm must learn to predict the future given only a partial temporal context. This makes anticipation challenging and also differentiates it from activity recognition. Previous works treat anticipation as a recognition problem [121, 163, 192] and train discriminative classifiers (such as SVM or CRF) on the complete temporal context. However, at test time these classifiers only observe a partial temporal context and make predictions within a filtering framework. We model anticipation with a recurrent architecture which unfolds through time. This lets us train a single classifier that learns to handle partial temporal context of varying lengths.

Furthermore, anticipation in robotics applications is challenging because the contextual information can come from multiple sensors with different data modalities. Examples include autonomous vehicles that reason from multiple sensors [6] or robots that jointly reason over perception and language instructions [158]. In such applications the way information from different sensors is fused is critical to the application's final performance. We therefore build an end-to-end deep learning architecture which jointly learns to anticipate and fuse information from different sensors.

4.6.1 RNN with LSTM units for anticipation

At the time of training, we observe the complete temporal observation sequence and the event $\{(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T), \mathbf{y}\}$. Our goal is to train a network which predicts the future event given a partial temporal observation sequence $\{(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t) | t < T\}$. We do so by training an RNN in a sequence-to-sequence prediction manner. Given training examples $\{(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)_j, \mathbf{y}_j\}_{j=1}^N$ we train an RNN with LSTM units to map the sequence of observations $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)$ to the sequence of events $(\mathbf{y}_1, ..., \mathbf{y}_T)$ such that $\mathbf{y}_t = \mathbf{y}, \forall t$, as shown in Figure 4.2. Trained in this manner, our RNN will attempt to map all sequences of partial observations $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t)$ $\forall t \leq T$ to the future event $\mathbf{y}$. This way our model explicitly learns to anticipate. We additionally use LSTM units which prevents the gradients from vanishing and allows our model to capture long temporal dependencies in human activities.[1]

[1]Driving maneuvers can take up to 6 seconds and the value of T can go up to 150 with a camera frame rate of 25 fps.

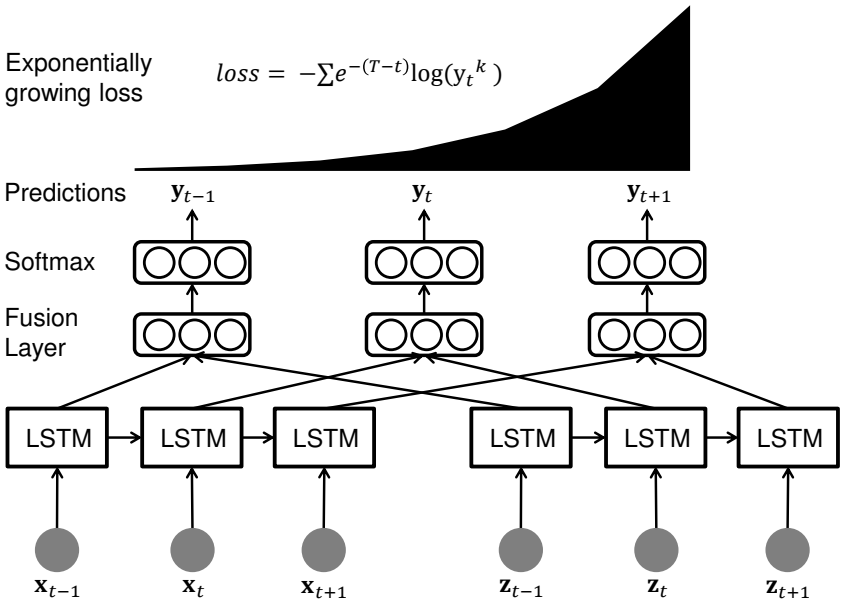


Figure 4.7: **Sensory fusion RNN for anticipation.** (**Bottom**) In the Fusion-RNN each sensory stream is passed through their independent RNN. (**Middle**) High-level representations from RNNs are then combined through a fusion layer. (**Top**) In order to prevent over-fitting early in time the loss exponentially increases with time.

4.6.2 Fusion-RNN: Sensory fusion RNN for anticipation

We now present an instantiation of our RNN architecture for fusing two sensory streams: $\{(\mathbf{x}_1, ..., \mathbf{x}_T),\ (\mathbf{z}_1, ..., \mathbf{z}_T)\}$. In the next section we will describe these streams for maneuver anticipation.

An obvious way to allow sensory fusion in the RNN is by concatenating the streams, i.e. using $([\mathbf{x}_1; \mathbf{z}_1], ..., [\mathbf{x}_T; \mathbf{z}_T])$ as input to the RNN. However, we found that this sort of simple concatenation performs poorly. We instead learn a sensory fusion layer which combines the high-level representations of sensor data. Our proposed architecture first passes the two sensory streams $\{(\mathbf{x}_1, ..., \mathbf{x}_T),\ (\mathbf{z}_1, ..., \mathbf{z}_T)\}$ independently through separate RNNs (4.9) and (4.10). The high level representations from both RNNs $\{(\mathbf{h}_1^x, ..., \mathbf{h}_T^x),\ (\mathbf{h}_1^z, ..., \mathbf{h}_T^z)$ are then

concatenated at each time step t and passed through a fully connected (fusion) layer which fuses the two representations (4.11), as shown in Figure 4.7. The output representation from the fusion layer is then passed to the softmax layer for anticipation (4.12). The following operations are performed from $t = 1$ to T.

$$(\mathbf{h}_t^x, \mathbf{c}_t^x) = \text{LSTM}_x(\mathbf{x}_t, \mathbf{h}_{t-1}^x, \mathbf{c}_{t-1}^x) \tag{4.9}$$

$$(\mathbf{h}_t^z, \mathbf{c}_t^z) = \text{LSTM}_z(\mathbf{z}_t, \mathbf{h}_{t-1}^z, \mathbf{c}_{t-1}^z) \tag{4.10}$$

$$\text{Sensory fusion: } \mathbf{e}_t = \texttt{tanh}(\mathbf{W}_f[\mathbf{h}_t^x; \mathbf{h}_t^z] + \mathbf{b}_f) \tag{4.11}$$

$$\mathbf{y}_t = \texttt{softmax}(\mathbf{W}_y \mathbf{e}_t + \mathbf{b}_y) \tag{4.12}$$

where $\mathbf{W}_*$ and $\mathbf{b}_*$ are model parameters, and LSTM_x and LSTM_z process the sensory streams $(\mathbf{x}_1, ..., \mathbf{x}_T)$ and $(\mathbf{z}_1, ..., \mathbf{z}_T)$ respectively. The same framework can be extended to handle more sensory streams.

4.6.3 Exponential loss-layer for anticipation.

We propose a new loss layer which encourages the architecture to anticipate early while also ensuring that the architecture does not over-fit the training data early enough in time when there is not enough context for anticipation. When using the standard softmax loss, the architecture suffers a loss of $-\log(y_t^k)$ for the mistakes it makes at each time step, where y_t^k is the probability of the ground truth event k computed by the architecture using Eq. (4.12). We propose to modify this loss by multiplying it with an exponential term as illustrated in Figure 4.7. Under this new scheme, the loss exponentially grows with time as shown below.

$$loss = \sum_{j=1}^{N} \sum_{t=1}^{T} -e^{-(T-t)} \log(y_t^k) \tag{4.13}$$

This loss penalizes the RNN exponentially more for the mistakes it makes as it sees more observations. This encourages the model to fix mistakes as early as it can in time. The loss in equation 4.13 also penalizes the network less on mistakes made early in time when there is not enough context available. This way it acts like a regularizer and reduces the risk to over-fit very early in time.

4.7 Features for anticipation

We extract features by processing the inside and outside driving contexts. We do this by grouping the overall contextual information from the sensors into: (i) the context from inside the vehicle, which comes from the driver facing camera and is represented as temporal sequence of features $(\mathbf{z}_1, ..., \mathbf{z}_T)$; and (ii) the context from outside the vehicle, which comes from the remaining sensors: GPS, road facing camera, and street maps. We represent the outside context with $(\mathbf{x}_1, ..., \mathbf{x}_T)$. In order to anticipate maneuvers, our RNN architecture (Figure 4.7) processes the temporal context $\{(\mathbf{x}_1, ..., \mathbf{x}_t), \ (\mathbf{z}_1, ..., \mathbf{z}_t)\}$ at every time step t, and outputs softmax probabilities $\mathbf{y}_t$ for the following five maneuvers: $\mathcal{M} = \{$*left turn*, *right turn*, *left lane change*, *right lane change*, *straight driving*$\}$.

4.7.1 Inside-vehicle features.

The inside features $\mathbf{z}_t$ capture the driver's head movements at each time instant t. Our vision pipeline consists of face detection, tracking, and feature extraction modules. We extract head motion features per-frame, denoted by ϕ(face). We compute $\mathbf{z}_t$ by aggregating ϕ(face) for every 20 frames, i.e., $\mathbf{z}_t$ =

$\sum_{i=1}^{20} \phi(\text{face}_i)/\| \sum_{i=1}^{20} \phi(\text{face}_i)\|$.

Face detection and tracking. We detect the driver's face using a trained Viola-Jones face detector [238]. From the detected face, we first extract visually discriminative (facial) points using the Shi-Tomasi corner detector [196] and then track those facial points using the Kanade-Lucas-Tomasi (KLT) tracker [148, 196, 227]. However, the tracking may accumulate errors over time because of changes in illumination due to the shadows of trees, traffic, etc. We therefore constrain the tracked facial points to follow a projective transformation and remove the incorrectly tracked points using the RANSAC algorithm. While tracking the facial points, we lose some of the tracked points with every new frame. To address this problem, we re-initialize the tracker with new discriminative facial points once the number of tracked points falls below a threshold [106].

Head motion features. For maneuver anticipation the horizontal movement of the face and its angular rotation (*yaw*) are particularly important. From the face tracking we obtain *face tracks*, which are 2D trajectories of the tracked facial points in the image plane. Figure 4.8 (bottom) shows how the horizontal coordinates of the tracked facial points vary with time before a left turn maneuver. We represent the driver's face movements and rotations with histogram features. In particular, we take matching facial points between successive frames and create histograms of their corresponding horizontal motions (in pixels) and angular motions in the image plane (Figure 4.8). We bin the horizontal and angular motions using $[\leq -2,\ -2 \text{ to } 0,\ 0 \text{ to } 2,\ \geq 2]$ and $[0 \text{ to } \frac{\pi}{2},\ \frac{\pi}{2} \text{ to } \pi,\ \pi \text{ to } \frac{3\pi}{2},\ \frac{3\pi}{2} \text{ to } 2\pi]$, respectively. We also calculate the mean movement of the driver's face center. This gives us $\phi(\text{face}) \in \mathbb{R}^9$ facial features per-frame. The driver's eye-gaze is also useful a feature. However, robustly estimating 3D eye-gaze in outside en-

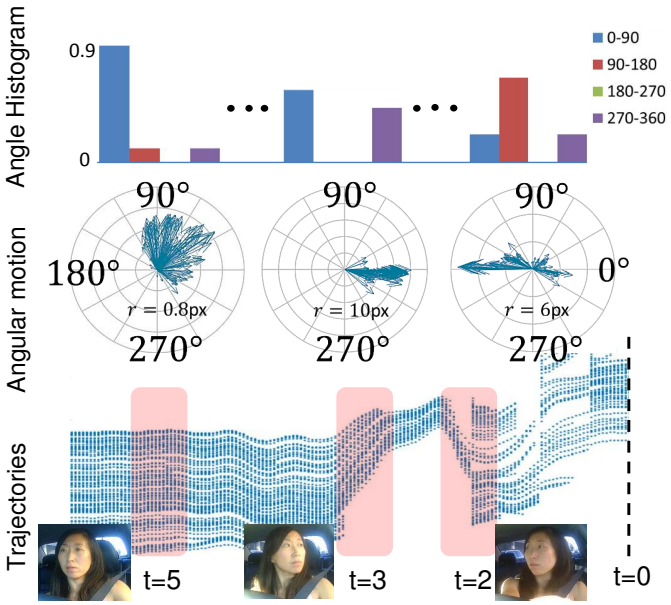


Figure 4.8: **Inside vehicle feature extraction.** The angular histogram features extracted at three different time steps for a left turn maneuver. *Bottom*: Trajectories for the horizontal motion of tracked facial pixels 't' seconds before the maneuver. At t=5 seconds before the maneuver the driver is looking straight, at t=3 looks (left) in the direction of maneuver, and at t=2 looks (right) in opposite direction for the crossing traffic. *Middle*: Average motion vector of tracked facial pixels in polar coordinates. *r* is the average movement of pixels and arrow indicates the direction in which the face moves when looking from the camera. *Top*: Normalized angular histogram features.

vironment is still a topic of research, and outside the scope of this chapter. We therefore do not consider eye-gaze features.

3D head pose and facial landmark features. Our framework is flexible and allows incorporating more advanced face detection and tracking algorithms. For example we replace the KLT tracker described above with the Constrained Local Neural Field (CLNF) model [9] and track 68 fixed landmark points on the driver's face. CLNF is particularly well suited for driving scenarios due its ability to handle a wide range of head pose and illumination variations. As shown

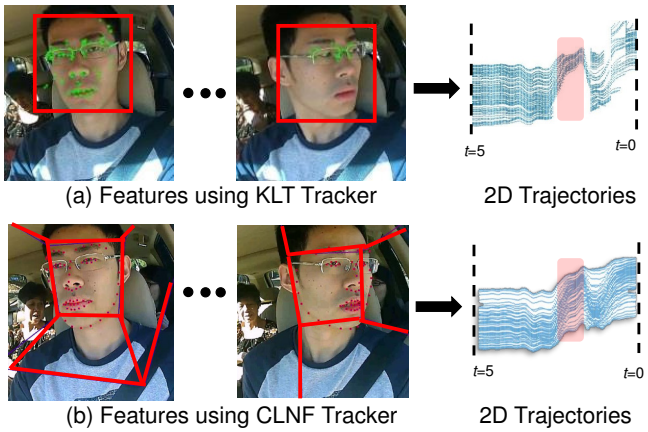


Figure 4.9: **Improved features for maneuver anticipation.** We track facial landmark points using the CLNF tracker [9] which results in more consistent 2D trajectories as compared to the KLT tracker [196] used by Jain et al. [91]. Furthermore, the CLNF also gives an estimate of the driver's 3D head pose.

in Figure 4.9, CLNF offers us two distinct benefits over the features from KLT (i) while discriminative facial points may change from situation to situation, tracking fixed landmarks results in consistent optical flow trajectories which adds to robustness; and (ii) CLNF also allows us to estimate the 3D head pose of the driver's face by minimizing error in the projection of a generic 3D mesh model of the face w.r.t. the 2D location of landmarks in the image. The histogram features generated from the optical flow trajectories along with the 3D head pose features (yaw, pitch and row), give us $\phi(\text{face}) \in \mathbb{R}^{12}$ when using the CLNF tracker.

In Section 4.9 we present results with the features from KLT, as well as the results with richer features obtained from the CLNF model.

4.7.2 Outside-vehicle features.

The outside feature vector $\mathbf{x}_t$ encodes the information about the outside environment such as the road conditions, vehicle dynamics, etc. In order to get this information, we use the road-facing camera together with the vehicle's GPS coordinates, its speed, and the street maps. More specifically, we obtain two binary features from the road-facing camera indicating whether a lane exists on the left side and on the right side of the vehicle. We also augment the vehicle's GPS coordinates with the street maps and extract a binary feature indicating if the vehicle is within 15 meters of a road artifact such as intersections, turns, highway exists, etc. We also encode the average, maximum, and minimum speeds of the vehicle over the last 5 seconds as features. This results in a $\mathbf{x}_t \in \mathbb{R}^6$ dimensional feature vector.

4.8 Bayesian networks for maneuver anticipation

In this section we propose alternate Bayesian networks [91] based on Hidden Markov Model (HMM) for maneuver anticipation. These models form a strong baseline to compare our sensory-fusion deep learning architecture.

Driving maneuvers are influenced by multiple interactions involving the vehicle, its driver, outside traffic, and occasionally global factors like the driver's destination. These interactions influence the driver's intention, i.e. their state of mind before the maneuver, which is not directly observable. In our Bayesian network formulation, we represent the driver's intention with discrete states that are *latent* (or hidden). In order to anticipate maneuvers, we jointly model

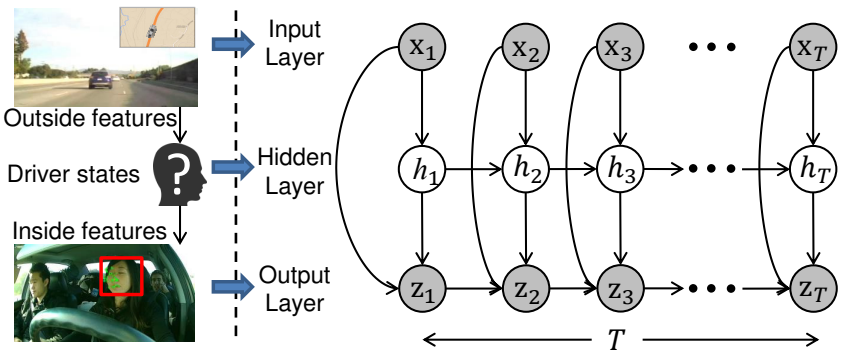


Figure 4.10: **AIO-HMM.** The model has three layers: (i) Input (top): this layer represents outside vehicle features $\mathbf{x}$; (ii) Hidden (middle): this layer represents driver's latent states h; and (iii) Output (bottom): this layer represents inside vehicle features $\mathbf{z}$. This layer also captures temporal dependencies of inside vehicle features. T represents time.

the driving context and the *latent* states in a tractable manner. We represent the driving context as a set of features described in Section 4.7. We now present the motivation for the Bayesian networks and then discuss our key model Autoregressive Input-Output HMM (AIO-HMM).

4.8.1 Modeling driving maneuvers

Modeling maneuvers require temporal modeling of the driving context. Discriminative methods, such as the Support Vector Machine and the Relevance Vector Machine [226], which do not model the temporal aspect perform poorly on anticipation tasks, as we show in Section 4.9. Therefore, a temporal model such as the Hidden Markov Model (HMM) is better suited to model maneuver anticipation.

An HMM models how the driver's *latent* states generate both the inside driving context ($\mathbf{z}_t$) and the outside driving context ($\mathbf{x}_t$). However, a more accurate

model should capture how events *outside* the vehicle (i.e. the outside driving context) affect the driver's state of mind, which then generates the observations *inside* the vehicle (i.e. the inside driving context). Such interactions can be modeled by an Input-Output HMM (IOHMM) [14]. However, modeling the problem with IOHMM does not capture the temporal dependencies of the inside driving context. These dependencies are critical to capture the smooth and temporally correlated behaviours such as the driver's face movements. We therefore present Autoregressive Input-Output HMM (AIO-HMM) which extends IOHMM to model these observation dependencies. Figure 4.10 shows the AIO-HMM graphical model for modeling maneuvers. We learn separate AIO-HMM model for each maneuver. In order to anticipate maneuvers, during inference we determine which model best explains the past several seconds of the driving context based on the data log-likelihood. In Appendix B.1 we describe the training and inference procedure for AIO-HMM.

4.9 Evaluation on real world driving data

In this section we first give an overview of our data set and then present the quantitative results. We also demonstrate our system and algorithm on real-world driving scenarios. Our video demonstrations are available at: `http://www.brain4cars.com`.

Figure 4.11: **Our data set** is diverse in drivers and landscape.

4.9.1 Driving data set

Our data set consists of natural driving videos with both inside and outside views of the car, its speed, and the global position system (GPS) coordinates.[2] The outside car video captures the view of the road ahead. We collected this driving data set under fully natural settings without any intervention.[3] It consists of 1180 miles of freeway and city driving and encloses 21,000 square miles across two states. We collected this data set from 10 drivers over a period of two months. The complete data set has a total of 2 million video frames and includes diverse landscapes. Figure 4.11 shows a few samples from our data set. We annotated the driving videos with a total of 700 events containing 274 lane changes, 131 turns, and 295 randomly sampled instances of driving straight. Each lane change or turn annotation marks the start time of the maneuver, i.e., before the car touches the lane or yaws, respectively. For all anno-

[2]The inside and outside cameras operate at 25 and 30 frames/sec.

[3]**Protocol:** We set up cameras, GPS and speed recording device in subject's personal vehicles and left it to record the data. The subjects were asked to ignore our setup and drive as they would normally.

tated events, we also annotated the lane information, i.e., the number of lanes on the road and the current lane of the car. Our data set is publicly available at `http://www.brain4cars.com`.

4.9.2 Baseline algorithms

We compare the following algorithms:

- *Chance*: Uniformly randomly anticipates a maneuver.
- *SVM* [163]: Support Vector Machine is a discriminative classifier [39]. Morris et al. [163] takes this approach for anticipating maneuvers.[4] We train the SVM on 5 seconds of driving context by concatenating all frame features to get a $\mathbb{R}^{3840}$ dimensional feature vector.
- *Random-Forest* [40]: This is also a discriminative classifier that learns many decision trees from the training data, and at test time it averages the prediction of the individual decision trees. We train it on the same features as SVM with 150 trees of depth ten each.
- *HMM*: This is the Hidden Markov Model. We train the HMM on a temporal sequence of feature vectors that we extract every 0.8 seconds, i.e., every 20 video frames. We consider three versions of the HMM: (i) HMM E: with only outside features from the road camera, the vehicle's speed, GPS and street maps (Section 4.7.2); (ii) HMM F: with only inside features from the driver's face (Section 4.7.1); and (ii) HMM $E+F$: with both inside and outside features.

[4] Morries et al. [163] considered binary classification problem (lane change vs driving straight) and used RVM [226].

- *IOHMM*: Jain et al. [91] modeled driving maneuvers with this Bayesian network. It is trained on the same features as HMM $E + F$.
- *AIO-HMM*: Jain et al. [91] proposed this Bayesian network for modeling maneuvers. It is trained on the same features as HMM $E + F$.
- *Simple-RNN*: In this architecture sensor streams are fused by simple concatenation and then passed through a single RNN with LSTM units.
- *Fusion-RNN-Uniform-Loss* (F-RNN-UL): In this architecture sensor streams are passed through separate RNNs, and the high-level representations from RNNs are then fused via a fully-connected layer. The loss at each time step takes the form $-\log(y_t^k)$.
- *Fusion-RNN-Exp-Loss* (F-RNN-EL): This architecture is similar to F-RNN-UL, except that the loss exponentially grows with time $-e^{-(T-t)}\log(y_t^k)$.

Our RNN and LSTM implementations are open-sourced and available at `NeuralModels` [89]. For the RNNs in our Fusion-RNN architecture we use a single layer LSTM of size 64 with sigmoid gate activations and tanh activation for hidden representation. Our fully connected fusion layer uses tanh activation and outputs a 64 dimensional vector. Our overall architecture (F-RNN-EL and F-RNN-UL) have nearly 25,000 parameters that are learned using RMSprop [44].

4.9.3 Evaluation protocol

We evaluate an algorithm based on its correctness in predicting future maneuvers. We anticipate maneuvers every 0.8 seconds where the algorithm processes the recent context and assigns a probability to each of the four maneuvers: {*left*

Algorithm 3: Maneuver anticipation

Initialize $m^* = \textit{driving straight}$

Input Features $\{(\mathbf{x}_1, ..., \mathbf{x}_T), (\mathbf{z}_1, ..., \mathbf{z}_T)\}$ and prediction threshold p_{th}

Output Predicted maneuver m^*

while $t = 1$ to T **do**

  Observe features $(\mathbf{x}_1, ..., \mathbf{x}_t)$ and $(\mathbf{z}_1, ..., \mathbf{z}_t)$

  Estimate probability $\mathbf{y}_t$ of each maneuver in $\mathcal{M}$

  $m_t^* = \arg\max_{m \in \mathcal{M}} \mathbf{y}_t$

  if $m_t^* \neq \textit{driving straight}$ & $\mathbf{y}_t\{m_t^*\} > p_{th}$ **then**

    $m^* = m_t^*$

    break

  end if

end while

Return m^*

lane change, right lane change, left turn, right turn} and a probability to the event of *driving straight*. These five probabilities together sum to one. After anticipation, i.e. when the algorithm has computed all five probabilities, the algorithm predicts a maneuver if its probability is above a threshold p_{th}. If none of the maneuvers' probabilities are above this threshold, the algorithm does not make a maneuver prediction and predicts *driving straight*. However, when it predicts one of the four maneuvers, it sticks with this prediction and makes no further predictions for next 5 seconds or until a maneuver occurs, whichever happens earlier. After 5 seconds or a maneuver has occurred, it returns to anticipating future maneuvers. Algorithm 3 shows the inference steps for maneuver anticipation.

During this process of anticipation and prediction, the algorithm makes (i) true predictions (tp): when it predicts the correct maneuver; (ii) false predictions (fp): when it predicts a maneuver but the driver performs a different maneuver; (iii) false positive predictions (fpp): when it predicts a maneuver but the driver does not perform any maneuver (i.e. *driving straight*); and (iv) missed predictions (mp): when it predicts *driving straight* but the driver performs a maneuver. We evaluate the algorithms using their precision and recall scores:

$$Pr = \frac{tp}{\underbrace{tp + fp + fpp}_{\text{Total \# of maneuver predictions}}}; \quad Re = \frac{tp}{\underbrace{tp + fp + mp}_{\text{Total \# of maneuvers}}}$$

The precision measures the fraction of the predicted maneuvers that are correct and recall measures the fraction of the maneuvers that are correctly predicted. For true predictions (tp) we also compute the average *time-to-maneuver*, where time-to-maneuver is the interval between the time of algorithm's prediction and the start of the maneuver.

We perform cross validation to choose the number of the driver's latent states in the AIO-HMM and the threshold on probabilities for maneuver prediction. For *SVM* we cross-validate for the parameter C and the choice of kernel from Gaussian and polynomial kernels. The parameters are chosen as the ones giving the highest F1-score on a validation set. The F1-score is the harmonic mean of the precision and recall, defined as $F1 = 2 * Pr * Re/(Pr + Re)$.

4.9.4 Quantitative results

We evaluate the algorithms on maneuvers that were not seen during training and report the results using 5-fold cross validation. Table 4.1 reports the preci-

Table 4.1: **Maneuver Anticipation Results.** Average *precision*, *recall* and *time-to-maneuver* are computed from 5-fold cross-validation. Standard error is also shown. Algorithms are compared on the features from Jain et al. [91].

	Lane change			Turns			All maneuvers		
Method	*Pr* (%)	*Re* (%)	Time-to-maneuver (s)	*Pr* (%)	*Re* (%)	Time-to-maneuver (s)	*Pr* (%)	*Re* (%)	Time-to-maneuver (s)
Chance	33.3	33.3	-	33.3	33.3	-	20.0	20.0	-
Morris et al. [163] SVM	73.7 ± 3.4	57.8 ± 2.8	2.40	64.7 ± 6.5	47.2 ± 7.6	2.40	43.7 ± 2.4	37.7 ± 1.8	1.20
Random-Forest	71.2 ± 2.4	53.4 ± 3.2	3.00	68.6 ± 3.5	44.4 ± 3.5	1.20	51.9 ± 1.6	27.7 ± 1.1	1.20
HMM *E*	75.0 ± 2.2	60.4 ± 5.7	3.46	74.4 ± 0.5	66.6 ± 3.0	4.04	63.9 ± 2.6	60.2 ± 4.2	3.26
HMM *F*	76.4 ± 1.4	75.2 ± 1.6	3.62	75.6 ± 2.7	60.1 ± 1.7	3.58	64.2 ± 1.5	36.8 ± 1.3	2.61
HMM *E* + *F*	80.9 ± 0.9	79.6 ± 1.3	3.61	73.5 ± 2.2	75.3 ± 3.1	4.53	67.8 ± 2.0	67.7 ± 2.5	3.72
IOHMM	81.6 ± 1.0	79.6 ± 1.9	3.98	77.6 ± 3.3	75.9 ± 2.5	4.42	74.2 ± 1.7	71.2 ± 1.6	3.83
(*Our final Bayesian network*) AIO-HMM	83.8 ± 1.3	79.2 ± 2.9	3.80	80.8 ± 3.4	75.2 ± 2.4	4.16	77.4 ± 2.3	71.2 ± 1.3	3.53
Simple-RNN	85.4 ± 0.7	86.0 ± 1.4	3.53	75.2 ± 1.4	75.3 ± 2.1	3.68	78.0 ± 1.5	71.1 ± 1.0	3.15
F-RNN-UL	**92.7** ± 2.1	84.4 ± 2.8	3.46	81.2 ± 3.5	78.6 ± 2.8	3.94	82.2 ± 1.0	75.9 ± 1.5	3.75
(*Our final deep architecture*) F-RNN-EL	88.2 ± 1.4	**86.0** ± 0.7	3.42	**83.8** ± 2.1	**79.9** ± 3.5	3.78	**84.5** ± 1.0	**77.1** ± 1.3	3.58

sion and recall scores under three settings: (i) *Lane change*: when the algorithms only predict for the left and right lane changes. This setting is relevant for highway driving where the prior probabilities of turns are low; (ii) *Turns*: when the algorithms only predict for the left and right turns; and (iii) *All maneuvers*: here the algorithms jointly predict all four maneuvers. All three settings include the instances of *driving straight*.

Table 4.1 compares the performance of the baseline anticipation algorithms, Bayesian networks, and the variants of our deep learning model. All algorithms in Table 4.1 use same feature vectors and KLT face tracker which ensures a fair comparison. As shown in the table, overall the best algorithm for maneuver anticipation is F-RNN-EL, and the best performing Bayesian network is AIO-HMM. F-RNN-EL significantly outperforms AIO-HMM in every setting. This improvement in performance is because RNNs with LSTM units are very expressive models with an internal memory. This allows them to model the much needed long temporal dependencies for anticipation. Additionally, unlike AIO-

HMM, F-RNN-EL is a discriminative model that does not make any assumptions about the generative nature of the problem. The results also highlight the importance of modeling the temporal nature in the data. Classifiers like SVM and Random Forest do not model the temporal aspects and hence performs poorly.

The performance of several variants of our deep architecture, reported in Table 4.1, justifies our design decisions to reach the final fusion architecture. When predicting all maneuvers, F-RNN-EL gives 6% higher precision and recall than Simple-RNN, which performs a simple fusion by concatenating the two sensor streams. On the other hand, F-RNN models each sensor stream with a separate RNN and then uses a fully connected layer to fuse the high-level representations at each time step. This form of sensory fusion is more principled since the sensor streams represent different data modalities. In addition, exponentially growing the loss further improves the performance. Our new loss scheme penalizes the network proportional to the length of context it has seen. When predicting all maneuvers, we observe that F-RNN-EL shows an improvement of 2% in precision and recall over F-RNN-UL. We conjecture that exponentially growing the loss acts like a regularizer. It reduces the risk of our network over-fitting early in time when there is not enough context available. Furthermore, the time-to-maneuver remains comparable for F-RNN with and without exponential loss.

The Bayesian networks AIO-HMM and HMM $E + F$ adopt different sensory fusion strategies. AIO-HMM fuses the two sensory streams using an input-output model, on the other hand HMM $E + F$ performs early fusion by concatenation. As a result, AIO-HMM gives 10% higher precision than HMM $E + F$ for jointly predicting all the maneuvers. AIO-HMM further extends IOHMM

Table 4.2: **False positive prediction** (fpp) of different algorithms. The number inside parenthesis is the standard error.

Algorithm	Lane change	Turns	All
Morris et al. [163] SVM	15.3 (0.8)	13.3 (5.6)	24.0 (3.5)
Random-Forest	16.2 (3.3)	12.9 (3.7)	17.5 (4.0)
HMM E	36.2 (6.6)	33.3 (0.0)	63.8 (9.4)
HMM F	23.1 (2.1)	23.3 (3.1)	11.5 (0.1)
HMM $E + F$	30.0 (4.8)	21.2 (3.3)	40.7 (4.9)
IOHMM	28.4 (1.5)	25.0 (0.1)	40.0 (1.5)
AIO-HMM	24.6 (1.5)	20.0 (2.0)	30.7 (3.4)
Simple-RNN	16.2 (1.3)	16.7 (0.0)	19.2 (0.0)
F-RNN-UL	19.2 (2.4)	25.0 (2.4)	21.5 (2.1)
F-RNN-EL	10.8 (0.7)	23.3 (1.5)	27.7 (3.8)

by modeling the temporal dependencies of events inside the vehicle. This results in better performance: on average AIO-HMM precision is 3% higher than IOHMM, as shown in Table 4.1. Another important aspect of anticipation is the joint modeling of the inside and outside driving contexts. HMM F learns only from the inside driving context, while HMM E learns only from the outside driving context. The performances of both the models is therefore less than HMM $E + F$, which learns jointly both the contexts.

Table 4.2 compares the fpp of different algorithms. False positive predictions (fpp) happen when an algorithm predicts a maneuver but the driver does not perform any maneuver (i.e. drives straight). Therefore low value of fpp is preferred. HMM F performs best on this metric at 11% as it mostly assigns a high probability to *driving straight*. However, due to this reason, it incorrectly predicts *driving straight* even when maneuvers happen. This results in the low recall of HMM F at 36%, as shown in Table 4.1. AIO-HMM's fpp is 10% less than that of IOHMM and HMM $E + F$, and F-RNN-EL is 3% less than AIO-

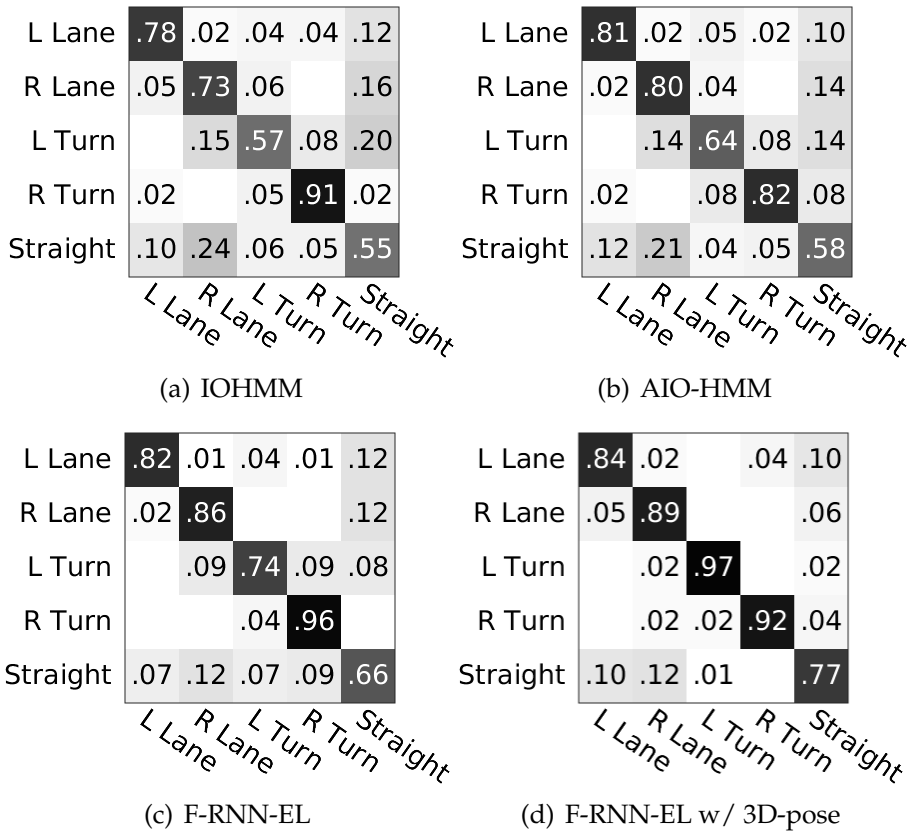
HMM. The primary reason for false positive predictions is distracted driving. Drivers interactions with fellow passengers or their looking at the surrounding scenes are sometimes wrongly interpreted by the algorithms. Understanding driver distraction is still an open problem, and outside the scope of this chapter.

Table 4.3: **3D head-pose features.** In this table we study the effect of better features with best performing algorithm from Table 4.1 in 'All maneuvers' setting. We use [9] to track 68 facial landmark points and estimate 3D head-pose.

Method	*Pr* (%)	*Re* (%)	Time-to-maneuver (s)
F-RNN-EL	84.5 (1.0)	77.1 (1.3)	3.58
F-RNN-EL w/ 3D head-pose	**90.5** (1.0)	**87.4** (0.5)	3.16

3D head-pose features. The modularity of our approach allows experimenting with more advanced head tracking algorithms. We replace the pipeline for extracting features from the driver's face [91] by a Constrained Local Neural Field (CLNF) model [9]. The new vision pipeline tracks 68 facial landmark points and estimates the driver's 3D head pose as described in Section 4.7. As shown in Table 4.3, we see a significant, 6% increase in precision and 10% increase in recall of F-RNN-EL when using features from our new vision pipeline. This increase in performance is attributed to the following reasons: (i) robustness of CLNF model to variations in illumination and head pose; (ii) 3D head-pose features are very informative for understanding the driver's intention; and (iii) optical flow trajectories generated by tracking facial landmark points represent head movements better, as shown in Figure 4.9.

The confusion matrix in Figure 4.12 shows the precision for each maneuver. F-RNN-EL gives a higher precision than AIO-HMM on every maneuver when both algorithms are trained on same features (Figure 4.12c). The new vision



(a) IOHMM (b) AIO-HMM

(c) F-RNN-EL (d) F-RNN-EL w/ 3D-pose

Figure 4.12: **Confusion matrix** of different algorithms when jointly predicting all the maneuvers. Predictions made by algorithms are represented by rows and actual maneuvers are represented by columns. Numbers on the diagonal represent precision.

pipeline with CLNF tracker further improves the precision of F-RNN-EL on all maneuvers (Figure 4.12d).

Effect of prediction threshold. In Figure 4.13 we study how F1-score varies as we change the prediction threshold p_{th}. We make the following observations: (i) The F1-score does not undergo large variations with changes to the prediction threshold. Hence, it allows practitioners to fairly trade-off between the precision and recall without hurting the F1-score by much; and (ii) the maximum F1-score attained by F-RNN-EL is 4% more than AIO-HMM when compared on the same features and 13% more with our new vision pipeline. In Tables 4.1, 4.2 and 4.3,

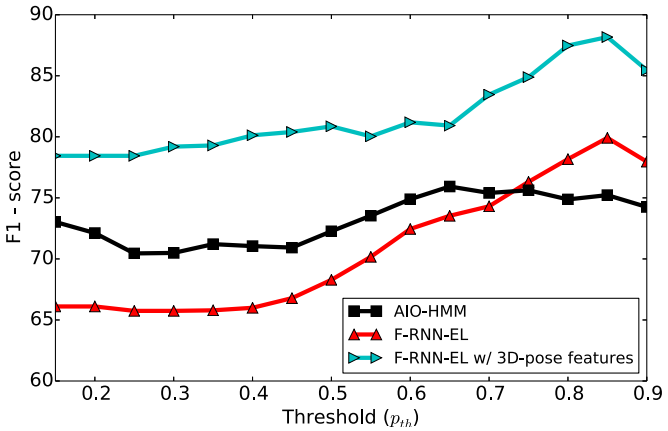


Figure 4.13: **Effect of prediction threshold p_{th}.** At test time an algorithm makes a prediction only when it is at least p_{th} confident in its prediction. This plot shows how F1-score vary with change in prediction threshold.

we used the threshold values which gave the highest F1-score.

Anticipation complexity. The F-RNN-EL anticipates maneuvers every 0.8 seconds using the previous 5 seconds of the driving context. The complexity mainly comprises of feature extraction and the model inference in Algorithm 3. Fortunately both these steps can be performed as a dynamic program by storing the computation of the most recent anticipation. Therefore, for every anticipation we only process the incoming 0.8 seconds and not complete 5 seconds of the driving context. On average we predict a maneuver under 0.20 milliseconds using Theano [10] on Nvidia K40 GPU on Ubuntu 12.04.

4.10 Conclusion

In this chapter we considered the problem of anticipating driving maneuvers a few seconds before the driver performs them. This problem requires model-

ing of long temporal interactions between a car, its driver, and the surrounding environment. We proposed a novel deep learning architecture based on Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) units for anticipation. Our architecture learns to fuse multiple sensory streams, and by training it in a sequence-to-sequence prediction manner, it explicitly learns to anticipate using only a partial temporal context. We also proposed a novel loss layer for anticipation which prevents over-fitting.

We release an open-source data set of 1180 miles of natural driving. We performed an extensive evaluation and showed improvement over many baseline algorithms. Our sensory fusion deep learning approach gives a precision of 84.5% and recall of 77.1%, and anticipates maneuvers 3.5 seconds (on average) before they happen. By incorporating the driver's 3D head-pose our precision and recall improves to 90.5% and 87.4% respectively. Potential application of our work is enabling advanced driver assistance systems (ADAS) to alert drivers before they perform a dangerous maneuver, thereby giving drivers more time to react. We believe that our deep learning architecture is widely applicable to many activity anticipation problems.

CHAPTER 5

DEEP LEARNING ON SPATIO-TEMPORAL GRAPHS

The interactions we discussed in the previous chapters – between human, robot, and their surrounding – were spanned over both space and time. Such spatio-temporal interactions are frequently encountered in computer vision and robotics, for example human-object interaction during an activity. These interactions are commonly represented over spatio-temporal graphs and modeled using Conditional Random Fields [133] or Bayesian Networks [164]. In this chapter we propose a novel framework for modeling spatio-temporal interactions using deep architectures.

Deep Recurrent Neural Network architectures, though remarkably capable at modeling sequences, lack an intuitive high-level spatio-temporal structure. That is while many problems in robotics inherently have an underlying high-level structure and can benefit from it. Spatio-temporal graphs are a popular flexible tool for imposing such high-level intuitions in the formulation of real world problems. In this chapter, we propose an approach for combining the power of high-level spatio-temporal graphs and sequence learning success of Recurrent Neural Networks (RNNs). We develop a scalable method for casting an arbitrary spatio-temporal graph as a rich RNN mixture that is feedforward, fully differentiable, and jointly trainable. The proposed method is generic and principled as it can be used for transforming any spatio-temporal graph through employing a certain set of well defined steps. We evaluate the proposed approach on a diverse set of problems, ranging from modeling human motion to object interactions, maneuver anticipation, and show improvement over the state-of-the-art with a large margin.

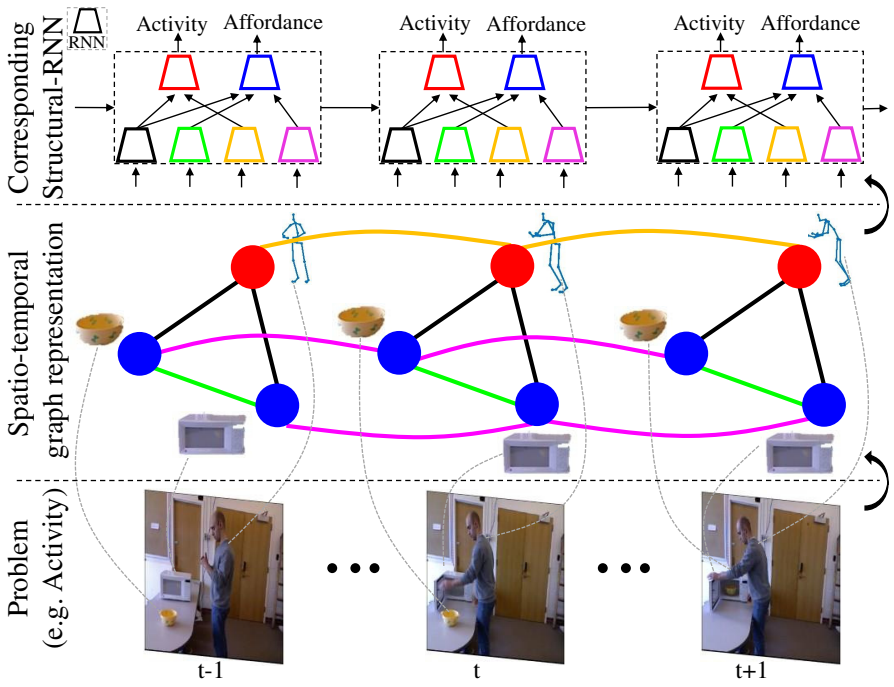


Figure 5.1: **From st-graph to S-RNN for an example problem. (Bottom)** Shows an example activity (human microwaving food). Modeling such problems requires both spatial and temporal reasoning. **(Middle)** St-graph capturing spatial and temporal interactions between the human and the objects. **(Top)** Schematic representation of our structural-RNN architecture automatically derived from st-graph. It captures the structure and interactions of st-graph in a rich yet scalable manner.

5.1 Recurrent neural networks and spatio-temporal graphs

The world we live in is inherently structured. It is comprised of components that interact with each other in space and time, leading to a spatio-temporal composition. Utilizing such structures in problem formulation allows domain-experts to inject their high-level knowledge in learning frameworks. This has been the incentive for many efforts in computer vision and machine learning, such as Logic Networks [186], Graphical Models [118], and Structured SVMs [104]. Structures that span over both space and time (spatio-temporal) are of particu-

lar interest to computer vision and robotics communities. Primarily, interactions between humans and environment in real world are inherently spatio-temporal in nature. For example, during a cooking activity, humans interact with multiple objects both in space and through time. Similarly, parts of human body (arms, legs, etc.) have individual functions but work with each other in concert to generate physically sensible motions. Hence, bringing high-level spatio-temporal structures and rich sequence modeling capabilities together is of particular importance for many applications.

The notable success of RNNs has proven their capability on many end-to-end learning tasks [76, 67, 49, 253]. However, they lack a high-level and intuitive spatio-temporal structure though they have been shown to be successful at modeling long-linear sequences [207, 156, 214]. Therefore, augmenting a high-level structure with learning capability of RNNs leads to a powerful tool that has the best of both worlds. Spatio-temporal graphs (st-graphs) are a popular [144, 142, 26, 52, 122, 252, 91] general tool for representing such high-level spatio-temporal structures. The nodes of the graph typically represent the problem components, and the edges capture their spatio-temporal interactions. To achieve the above goal, we develop a generic tool for transforming an arbitrary st-graph into a feedforward mixture of RNNs, named structural-RNN (S-RNN). Figure 5.1 schematically illustrates this process, where a sample spatio-temporal problem is shown at the bottom, the corresponding st-graph representation is shown in the middle, and our RNN mixture counterpart of the st-graph is shown at the top.

In high-level steps, given an arbitrary st-graph, we first roll it out in time and decompose it into a set of contributing factor components. The factors iden-

tify the independent components that collectively determine one decision and are derived from both edges and nodes of the st-graph. We then semantically group the factor components and represent each group using one RNN, which results in the desired RNN mixture. The main challenges of this transformation problem are: 1) making the RNN mixture as rich as possible to enable learning complex functions, yet 2) keeping the RNN mixture scalable with respect to size of the input st-graph. In order to make the resulting RNN mixture rich, we liberally represent each spatio-temporal factor (including node factors, temporal edge factors, and spatio-temporal edge factors) using one RNN. On the other hand, to keep the overall mixture scalable but not lose the essential learning capacity, we utilize "factor sharing" (aka clique templates [217, 154, 215]) and allow the factors with similar semantic functions to share an RNN. This results in a rich and scalable feedforward mixture of RNNs that is equivalent to the provided st-graph in terms of input, output, and spatio-temporal relationships. The mixture is also fully differentiable, and therefore, can be trained jointly as one entity.

The proposed method is principled and generic as the transformation is based on a set of well defined steps and it is applicable to any problem that can be formulated as st-graphs (as defined in Section 5.3). Several previous works have attempted solving specific problems using a collection of RNNs [207, 57, 236, 49, 27], but they are almost unanimously task-specific. They also do not utilize mechanisms similar to factorization or factor sharing in devising their architecture to ensure richness and scalability.

Our approach allows domain-experts to cast their problems as st-graphs and learn deep recurrent structures over them. S-RNN is also modular, as it is en-

joying an underlying high-level structure. This enables easy high-level manipulations which are basically not possible in unstructured (plain-vanilla) RNNs (e.g., we will experimentally show forming a feasible hybrid human motion by mixing parts of different motion styles - Sec 5.4.2). We evaluate the proposed approach on a diverse set of spatio-temporal problems (human pose modeling and forecasting, human-object interaction, and driver decision making), and show significant improvements over the state of the art on each problem. We also study complexity and convergence properties of S-RNN and provide further experimental insights by visualizing its memory cells that reveals some cells interestingly represent certain semantic operations. The code of the entire framework that accepts a st-graph as the input and yields the output RNN mixture is available at the `http://asheshjain.org/srnn`

The chapter contributes: 1) a generic method for casting an arbitrary st-graph as a rich, scalable, and jointly trainable RNN mixture, 2) in defence of structured approaches, we show S-RNN significantly outperforms its unstructured (plain-vanilla) RNN counterparts, 3) in defence of RNNs, we show on several diverse spatio-temporal problems that modeling structure with S-RNN outperforms the non-deep learning based structured counterparts.

5.2 Related deep architectures

We give a categorized overview of the related literature. In general, three main characteristics differentiate our work from existing techniques: being generic and not restricted to a specific problem, providing a principled method for transforming a st-graph into a scalable rich feedforward RNN mixture, and be-

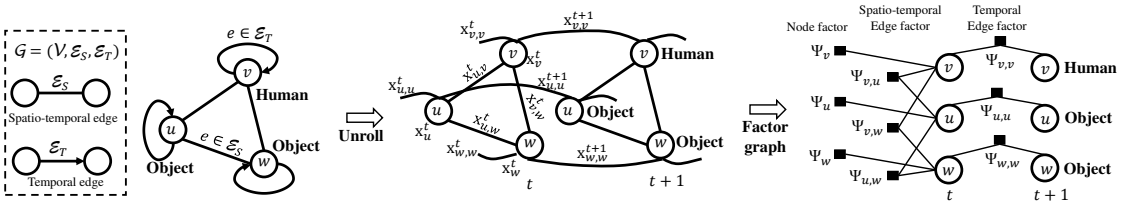
ing jointly trainable.

Spatio-temporal problems. Problems that require spatial and temporal reasoning are very common in robotics and computer vision. Examples include human activity recognition and segmentation from videos [211, 197, 240, 234, 33, 98, 142, 135], context-rich human-object interactions [144, 123, 119, 79], modeling human motion [67, 221, 220] etc. Spatio-temporal reasoning also finds application in assistive cars, driver understanding, and multi-sensor object recognition [252, 91, 176, 52]. In fact most of our daily activities are spatio-temporal in nature. With growing interests in rich interactions and robotics, this form of reasoning will become even more important. We evaluate our generic method on three context-rich spatio-temporal problems: (i) Human motion modeling [67]; (ii) Human-object interaction understanding [123]; and (iii) Driver maneuver anticipation [91].

Mixtures of deep architectures. Several previous works build multiple networks and wire them together in order to capture some complex structure (or interactions) in the problem with promising results on applications such as activity detection, scene labeling, image captioning, and object detection [57, 27, 35, 72, 207, 236]. However, such architectures are mostly hand designed for specific problems, though they demonstrate the benefit in using a modular deep architecture. Recursive Neural Networks [73] are, on the other hand, generic feedforward architectures, but for problems with recursive structure such as parsing of natural sentences and scenes [205]. Our work is a remedy for problems expressed as spatio-temporal graphs. For a new spatio-temporal problem in hand, all a practitioner needs to do is to express their intuition about the problem as an st-graph.

Deep learning with graphical models. Many works have addressed deep networks with graphical models for structured prediction tasks. Bengio et al. [15] combined CNNs with HMM for hand writing recognition. Tompson et al. [228] jointly train CNN and MRF for human pose estimation. Chen et al. [32] use a similar approach for image classification with general MRF. Recently several works have addressed end-to-end image segmentation with fully connected CRF [253, 147, 69, 146]. Several works follow a two-stage approach and decouple the deep network from CRF. They have been applied to multiple problems including image segmentation, pose estimation, document processing [251, 31, 143, 25] etc. All of these works advocate and well demonstrate the benefit in exploiting the structure in the problem together with rich deep architectures. However, they largely do not address spatio-temporal problems and the proposed architectures are task-specific.

Conditional Random Fields (CRF) model dependencies between the outputs by learning a joint distribution over them. They have been applied to many applications [127, 62, 177] including st-graphs which are commonly modeled as spatio-temporal CRF [144, 123, 252, 52]. In our approach, we adopt st-graphs as a general graph representation and embody it using an RNN mixture architecture. Unlike CRF, our approach is not probabilistic and is not meant to model the joint distribution over the outputs. S-RNN instead learns the dependencies between the outputs via structural sharing of RNNs between the outputs.



(a) Spatio-temporal graph representing an activity (b) Unrolled through time (c) Factor graph parameterization

Figure 5.2: **An example spatio-temporal graph (st-graph) of a human activity.** (a) st-graph capturing human-object interaction. (b) Unrolling the st-graph through edges $\mathcal{E}_T$. The nodes and edges are labelled with the feature vectors associated with them. (c) Our factor graph parameterization of the st-graph. Each node and edge in the st-graph has a corresponding factor.

5.3 Structural-RNN architectures

In this section we describe our approach for building structural-RNN (S-RNN) architectures. We start with a st-graph, decompose it into a set of factor components, then represent each factor using a RNN. The RNNs are interconnected in a way that the resulting architecture captures the structure and interactions of the st-graph.

5.3.1 Representation of spatio-temporal graphs

Many applications that require spatial and temporal reasoning are modeled using st-graphs [26, 52, 123, 252, 91]. We represent a st-graph with $\mathcal{G} = (\mathcal{V}, \mathcal{E}_S, \mathcal{E}_T)$, whose structure $(\mathcal{V}, \mathcal{E}_S)$ unrolls over time through edges $\mathcal{E}_T$. Figure 5.2a shows an example st-graph capturing human-object interactions during an activity. Figure 5.2b shows the same st-graph unrolled over time. Note that the nodes $v \in \mathcal{V}$ and edges $e \in \mathcal{E}_S \cup \mathcal{E}_T$ of the st-graph are *temporal* in nature. At

each time step t, the nodes are connected with undirected *spatio-temporal* edge $e = (u, v) \in \mathcal{E}_S$. The node u at time t and the node v at time $t + 1$ are connected with an undirected *temporal* edge *iff* $(u, v) \in \mathcal{E}_T$.[1]

Given a st-graph and the feature vectors associated with the nodes $\mathbf{x}_v^t$ and edges $\mathbf{x}_e^t$, as shown in Figure 5.2b, the goal is to predict the node labels (or real value vectors) y_v^t at each time step t. For instance, in human-object interaction, the node features can represent the human and object poses, and edge features can their relative orientation; the node labels represent the human activity and object affordance. Label y_v^t is affected by both its node and its interactions with other nodes (edges), leading to an overall complex system. Such interactions are commonly parameterized with a factor graph that conveys how a (complicated) function over the st-graph factorizes into simpler functions [129]. We derive our S-RNN architecture from the factor graph representation of the st-graph. Our factor graph representation has a factor function $\Psi_v(y_v, \mathbf{x}_v)$ for each node and a pairwise factor $\Psi_e(y_{e(1)}, y_{e(2)}, \mathbf{x}_e)$ for each edge. Figure 5.2c shows the factor graph corresponding to the st-graph in 5.2a. [2]

Sharing factors between nodes. Each factor in the st-graph has parameters that needs to be learned. Instead of learning a distinct factor for each node, semantically similar nodes can optionally share factors. For example, all "object nodes" $\{u,w\}$ in the st-graph can share the same node factor and parameters. This modeling choice allows enforcing parameter sharing between similar nodes. It further gives the flexibility to handle st-graphs with more nodes with-

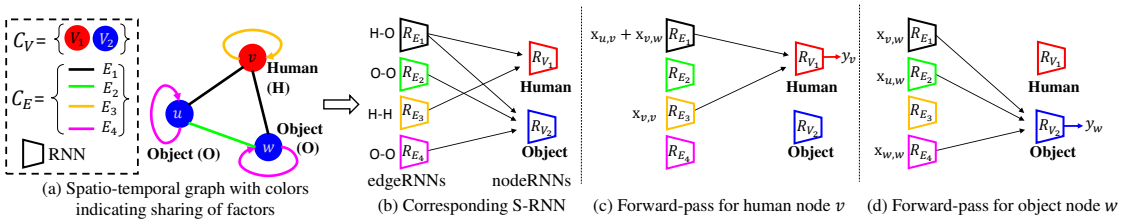
[1]For simplicity, the example st-graph in Figure 5.2a considers temporal edges of the form $(v, v) \in \mathcal{E}_T$.

[2]Note that we adopted factor graph as a tool for capturing interactions and not modeling the overall function. Factor graphs are commonly used in probabilistic graphical models for factorizing joint probability distributions. We consider them for general st-graphs and do not establish relations to its probabilistic and function decomposition properties.

out increasing the number of parameters. For this purpose, we partition the nodes as $C_V = \{V_1, .., V_P\}$ where V_p is a set of semantically similar nodes, and they all use the same node factor Ψ_{V_p}. In Figure 5.3a we re-draw the st-graph and assign same color to the nodes sharing node factors.

Partitioning nodes on their semantic meanings leads to a natural semantic partition of the edges, $C_E = \{E_1, .., E_M\}$, where E_m is a set of edges whose nodes form a semantic pair. Therefore, all edges in the set E_m share the same edge factor Ψ_{E_m}. For example all "human-object edges" $\{(v, u), (v, w)\}$ are modeled with the same edge factor. Sharing factors based on semantic meaning makes the overall parametrization compact. In fact, sharing parameters is necessary to address applications where the number of nodes depends on the context. For example, in human-object interaction the number of object nodes vary with the environment. Therefore, without sharing parameters between the object nodes, the model cannot generalize to new environments with more objects. For modeling flexibility, the edge factors are not shared across the edges in $\mathcal{E}_S$ and $\mathcal{E}_T$. Hence, in Figure 5.3a, object-object $(w, w) \in \mathcal{E}_T$ temporal edge is colored differently from object-object $(u, w) \in \mathcal{E}_S$ spatio-temporal edge.

In order to predict the label of node $v \in V_p$, we consider its node factor Ψ_{V_p}, and the edge factors connected to v in the factor graph. We define a node factor and an edge factor as neighbors if they jointly affect the label of some node in the st-graph. More formally, the node factor Ψ_{V_p} and edge factor Ψ_{E_m} are *neighbors*, if there exist a node $v \in V_p$ such that it connects to both Ψ_{V_p} and Ψ_{E_m} in the factor graph. We will use this definition in building S-RNN architecture such that it captures the interactions in the st-graph.



(a) Spatio-temporal graph with colors indicating sharing of factors

(b) Corresponding S-RNN

(c) Forward-pass for human node v

(d) Forward-pass for object node w

Figure 5.3: **An example of st-graph to S-RNN. (a)** The st-graph from Figure 5.2 is redrawn with colors to indicate sharing of nodes and edge factors. Nodes and edges with same color share factors. Overall there are six distinct factors: 2 node factors and 4 edge factors. **(b)** S-RNN architecture has one RNN for each factor. EdgeRNNs and nodeRNNs are connected to form a bipartite graph. Parameter sharing between the human and object nodes happen through edgeRNN $\mathbf{R}_{E_1}$. **(c)** The forward-pass for human node v involve RNNs $\mathbf{R}_{E_1}$, $\mathbf{R}_{E_3}$ and $\mathbf{R}_{V_1}$. In Figure 5.4 we show the detailed layout of this forward-pass. Input features into $\mathbf{R}_{E_1}$ is sum of human-object edge features $\mathbf{x}_{u,v} + \mathbf{x}_{v,w}$. **(d)** The forward-pass for object node w involve RNNs $\mathbf{R}_{E_1}$, $\mathbf{R}_{E_2}$, $\mathbf{R}_{E_4}$ and $\mathbf{R}_{V_2}$. In this forward-pass, the edgeRNN $\mathbf{R}_{E_1}$ only processes the edge feature $\mathbf{x}_{v,w}$. (Best viewed in color)

5.3.2 Structural-RNN from spatio-temporal graphs

We derive our S-RNN architecture from the factor graph representation of the st-graph. The factors in the st-graph operate in a temporal manner, where at each time step the factors observe (node & edge) features and perform some computation on those features. In S-RNN, we represent each factor with an RNN. We refer the RNNs obtained from the node factors as nodeRNNs and the RNNs obtained from the edge factors as edgeRNNs. The interactions represented by the st-graph are captured through connections between the nodeRNNs and the edgeRNNs.

We denote the RNNs corresponding to the node factor Ψ_{V_p} and the edge factor Ψ_{E_m} as $\mathbf{R}_{V_p}$ and $\mathbf{R}_{E_m}$ respectively. In order to obtain a feedforward net-

Algorithm 4: From spatio-temporal graph to S-RNN

Input $\mathcal{G} = (\mathcal{V}, \mathcal{E}_S, \mathcal{E}_T)$, $C_V = \{V_1, ..., V_P\}$

Output S-RNN graph $\mathcal{G}_R = (\{\mathbf{R}_{E_m}\}, \{\mathbf{R}_{V_p}\}, \mathcal{E}_R)$

1: Semantically partition edges $C_E = \{E_1, ..., E_M\}$

2: Find factor components $\{\Psi_{V_p}, \Psi_{E_m}\}$ of $\mathcal{G}$

3: Represent each Ψ_{V_p} with a nodeRNN $\mathbf{R}_{V_p}$

4: Represent each Ψ_{E_m} with an edgeRNN $\mathbf{R}_{E_m}$

5: Connect $\{\mathbf{R}_{E_m}\}$ and $\{\mathbf{R}_{V_p}\}$ to form a bipartite graph.

$(\mathbf{R}_{E_m}, \mathbf{R}_{V_p}) \in \mathcal{E}_R$ *iff* $\exists v \in V_p, u \in \mathcal{V}$ s.t. $(u, v) \in E_m$

Return $\mathcal{G}_R = (\{\mathbf{R}_{E_m}\}, \{\mathbf{R}_{V_p}\}, \mathcal{E}_R)$

work, we connect the edgeRNNs and nodeRNNs to form a bipartite graph $\mathcal{G}_R = (\{\mathbf{R}_{E_m}\}, \{\mathbf{R}_{V_p}\}, \mathcal{E}_R)$. In particular, the edgeRNN $\mathbf{R}_{E_m}$ is connected to the nodeRNN $\mathbf{R}_{V_p}$ *iff* the factors Ψ_{E_m} and Ψ_{V_p} are *neighbors* in the st-graph, i.e. they jointly affect the label of some node in the st-graph. To summarize, in Algorithm 4 we show the steps for constructing S-RNN architecture. Figure 5.3b shows the S-RNN for the human activity represented in Figure 5.3a. The nodeRNNs combine the outputs of the edgeRNNs they are connected to (i.e. its *neighbors* in the factor graph), and predict the node labels. The predictions of nodeRNNs (eg. $\mathbf{R}_{V_1}$ and $\mathbf{R}_{V_2}$) interact through the edgeRNNs (eg. $\mathbf{R}_{E_1}$). Each edgeRNN handles a specific semantic interaction between the nodes connected in the st-grap and models how the interactions evolve over time. In the next section, we explain the inputs, outputs, and the training procedure of S-RNN.

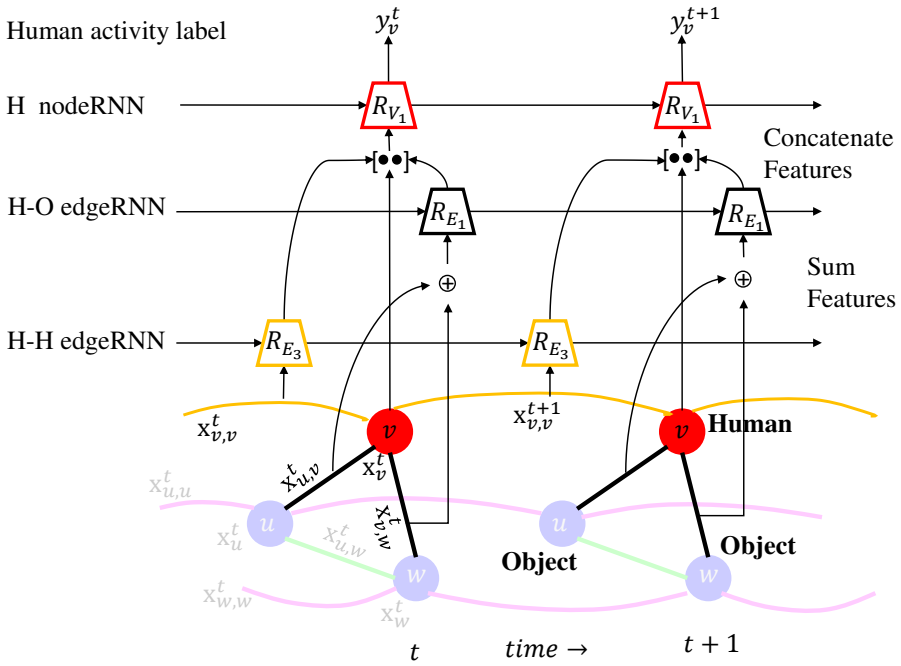


Figure 5.4: **Forward-pass for human node** v**.** Shows the architecture layout corresponding to the Figure 5.3c on unrolled st-graph. (View in color)

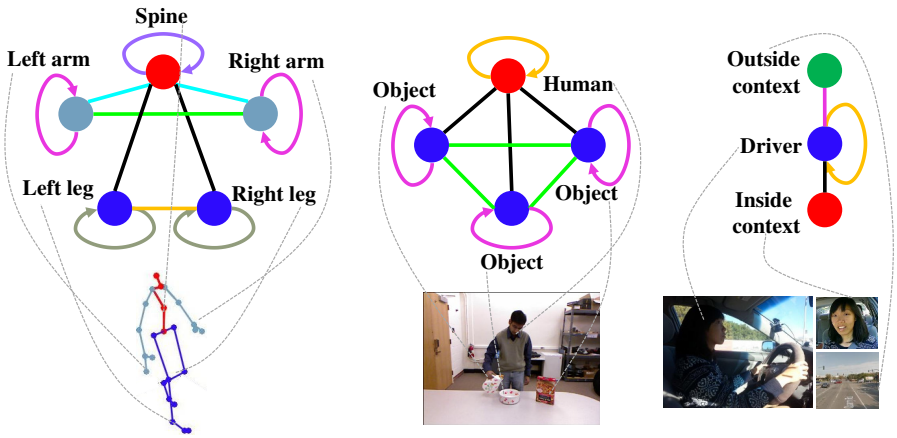
5.3.3 Training structural-RNN architecture

In order to train the S-RNN architecture, for each node of the st-graph the features associated with the node are fed into the architecture. In the forward-pass for node $v \in V_p$, the input into edgeRNN $\mathbf{R}_{E_m}$ is the temporal sequence of edge features $\mathbf{x}_e^t$ on the edge $e \in E_m$, where edge e is incident to node v in the st-graph. The nodeRNN $\mathbf{R}_{V_p}$ at each time step concatenates the node feature $\mathbf{x}_v^t$ and the outputs of edgeRNNs it is connected to, and predicts the node label. At the time of training, the errors in prediction are back-propagated through the nodeRNN and edgeRNNs involved during the forward-pass. That way, S-RNN non-linearly combines the node and edge features associated with the nodes in order to predict the node labels.

Figure 5.3c shows the forward-pass through S-RNN for the human node. Figure 5.4 shows a detailed architecture layout of the same forward-pass. The forward-pass involves the edgeRNNs $\mathbf{R}_{E_1}$ (human-object edge) and $\mathbf{R}_{E_3}$ (human-human edge). Since the human node v interacts with two object nodes $\{u,w\}$, we pass the summation of the two edge features as input to $\mathbf{R}_{E_1}$. The summation of features, as opposed to concatenation, is important to handle variable number of object nodes with a fixed architecture. Since the object count varies with environment, it is challenging to represent variable context with a fixed length feature vector. Empirically, adding features works better than mean pooling. We conjecture that addition retains the object count and the structure of the st-graph, while mean pooling averages out the number of edges. The nodeRNN $\mathbf{R}_{V_1}$ concatenates the (human) node features with the outputs from edgeRNNs, and predicts the activity at each time step.

Parameter sharing and structured feature space. An important aspect of S-RNN is sharing of parameters across the node labels. Parameter sharing between node labels happen when an RNN is common in their forward-pass. For example in Figure 5.3c and 5.3d, the edgeRNN $\mathbf{R}_{E_1}$ is common in the forward-pass for the human node and the object nodes. Furthermore, the parameters of $\mathbf{R}_{E_1}$ gets updated through back-propagated gradients from both the object and human nodeRNNs. In this way, $\mathbf{R}_{E_1}$ affects both the human and object node labels.

Since the human node is connected to multiple object nodes, the input into edgeRNN $\mathbf{R}_{E_1}$ is always a linear combination of human-object edge features. This imposes an structure on the features processed by $\mathbf{R}_{E_1}$. More formally, the input into $\mathbf{R}_{E_1}$ is the inner product $\mathbf{s}^T\mathbf{F}$, where $\mathbf{F}$ is the feature matrix storing



(a) Human motion modeling (b) Activity detection and anticipation (c) Maneuver anticipation

Figure 5.5: **Diverse spatio-temporal tasks**. We apply S-RNN to the following three diverse spatio-temporal problems. (View in color)

the edge features $\mathbf{x}_e$ s.t. $e \in E_1$. Vector $\mathbf{s}$ captures the structured feature space. Its entries are in $\{0,1\}$ depending on the node being forward-passed. In the example above $\mathbf{F} = [\mathbf{x}_{v,u} \quad \mathbf{x}_{v,w}]^T$. For the human node v, $\mathbf{s} = [1\ 1]^T$, while for the object node u, $\mathbf{s} = [1\ 0]^T$.

5.4 Applications of Structural-RNN

We present results on three diverse spatio-temporal problems to ensure generic applicability of S-RNN, shown in Figure 5.5. The applications include: (i) modeling human motion [67] from motion capture data [86]; (ii) human activity detection and anticipation [119, 121]; and (iii) maneuver anticipation from real-world driving data [91].

5.4.1 Human motion modeling and forecasting

Human body is a good example of separate but well related components. Its motion involves complex spatio-temporal interactions between the components (arms, legs, spine), resulting in sensible motion styles like walking, eating etc. In this experiment, we represent the complex motion of humans over st-graphs and learn to model them with S-RNN. We show that our structured approach outperforms the state-of-the-art unstructured deep architecture [67] on motion forecasting from motion capture (mocap) data. Several approaches based on Gaussian processes [232, 241], Restricted Boltzmann Machines (RBMs) [221, 220, 213], and RNNs [67] have been proposed to model human motion. Recently, Fragkiadaki et al. [67] proposed an encoder-RNN-decoder (ERD) which gets state-of-the-art forecasting results on H3.6m mocap data set [86].

S-RNN architecture for human motion. Our S-RNN architecture follows the st-graph shown in Figure 5.5a. According to the st-graph, the spine interacts with all the body parts, and the arms and legs interact with each other. The st-graph is automatically transformed to S-RNN following Section 5.3.2. The resulting S-RNN have three nodeRNNs, one for each type of body part (spine, arm, and leg), four edgeRNNs for modeling the spatio-temporal interactions between them, and three edgeRNNs for their temporal connections. For edgeRNNs and nodeRNNs we use FC(256)-FC(256)-LSTM(512) and LSTM(512)-FC(256)-FC(100)-FC($\cdot$) architectures, respectively, with skip input and output connections [75]. The outputs of nodeRNNs are skeleton joints of different body parts, which are concatenated to reconstruct the complete skeleton. In order to model human motion, we train S-RNN to predict the mocap frame at time $t+1$ given the frame at time t. Similar to [67], we gradually add noise to the mocap frames

during training. This simulates curriculum learning [16] and helps in keeping the forecasted motion close to the manifold of human motion. As node features we use the raw joint values expressed as exponential map [67], and edge features are concatenation of the node features. We train all RNNs jointly to minimize the Euclidean loss between the predicted mocap frame and the ground truth. We closely follow the training procedure by Fragkiadaki et al. [67]. We cross-validate over the hyperparameters on the validation set and set them to the following values:

- Back propagation through 100 time steps.
- Each mini-batch have 100 sequences.
- We use SGD and start with the step-size of 10^{-3}. We decay the step-size by 0.1 when the training error plateaus.
- We clip the L2-norm of gradient to 25.0, and clip each dimension to [-5.0, 5.0]
- We gradually add Gaussian noise to the training data following the schedule: at iterations {250, 500, 1000, 1300, 2000, 2500, 3300} we add noise with standard deviation {0.01, 0.05, 0.1, 0.2, 0.3, 0.5, 0.7}. As also noted by Fragkiadaki et al. [67], adding noise is very important during training in order to forecast motions that lie on the manifold of human-like motions.

Evaluation setup. We compare S-RNN with the state-of-the-art ERD architecture [67] on H3.6m mocap data set [86]. We also compare with a 3 layer LSTM architecture (LSTM-3LR) which [67] use as a baseline.[3] For motion forecasting we follow the experimental setup of [67]. We downsample H3.6m by two and

[3]We reproduce ERD and LSTM-3LR architectures following [67]. The authors implementation were not available at the time of submission.

train on 6 subjects and test on subject S5. To forecast, we first feed the architectures with (50) *seed* mocap frames, and then forecast the future (100) frames. Following [67], we consider walking, eating, and smoking activities. In addition to these three, we also consider discussion activity.

Forecasting is specially challenging on activities with complex aperiodic human motion. In H3.6m data set, significant parts of eating, smoking, and discussion activities are aperiodic, while walking activity is mostly periodic. Our evaluation demonstrates the benefits of having an underlying structure in three important ways: (i) We present visualizations and quantitative results on complex aperiodic activities ([67] evaluates only on periodic walking motion); (ii) We forecast human motion for almost twice longer than state-of-the-art [67]. This is very challenging for aperiodic activities; and finally (iii) We show S-RNN interestingly learns semantic concepts, and demonstrate its modularity by generating hybrid human motion. Unstructured deep architectures like [67] does not offer such modularity.

Qualitative results on motion forecasting. Figure 5.6 shows forecasting 1000ms of human motion on "eating" activity – the subject drinks while walking. S-RNN stays close to the ground-truth in the short-term and generates human like motion in the long-term. On removing edgeRNNs, the parts of human body become independent and stops interacting through parameters. Hence without edgeRNNs the skeleton freezes to some mean position. LSTM-3LR suffers with a drifting problem. On many test examples it drifts to the mean position of walking human ([67] made similar observations about LSTM-3LR). The motion generated by ERD [67] stays human-like in the short-term but it drifts away to non-human like motion in the long-term. This was a common outcome of ERD

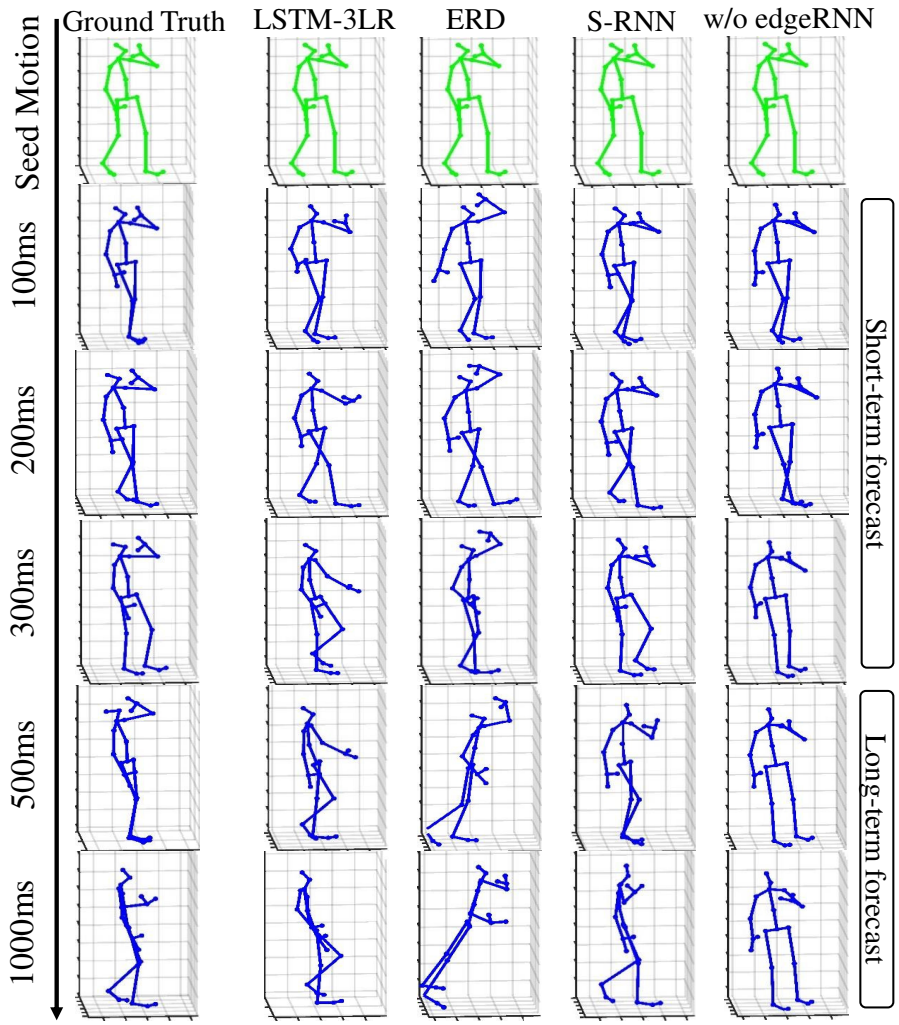


Figure 5.6: **Forecasting eating activity on test subject**. On aperiodic activities, ERD and LSTM-3LR struggle to model human motion. S-RNN, on the other hand, mimics the ground truth in the short-term and generates human like motion in the long term. Without (w/o) edgeRNNs the motion freezes to some mean standing position. See Video: `http://asheshjain.org/srnn`

on complex aperiodic activities, unlike S-RNN. Furthermore, ERD produced human motion was non-smooth on many test examples. See video for more examples: `http://asheshjain.org/srnn`

Quantitative evaluation. We follow the evaluation metric of Fragkiadaki et al. [67] and present the 3D angle error between the forecasted mocap frame and

Table 5.1: **Motion forecasting angle error**. {80, 160, 320, 560, 1000} msecs after the seed motion. The results are averaged over 8 seed motion sequences for each activity on the test subject.

Methods	Short-term forecast			Long-term forecast	
	80ms	160ms	320ms	560ms	1000ms
	Walking activity				
ERD [67]	1.30	1.56	1.84	2.00	2.38
LSTM-3LR	1.18	1.50	1.67	**1.81**	2.20
S-RNN	**1.08**	**1.34**	**1.60**	1.90	**2.13**
	Eating activity				
ERD [67]	1.66	1.93	2.28	2.36	**2.41**
LSTM-3LR	1.36	1.79	2.29	2.49	2.82
S-RNN	**1.35**	**1.71**	**2.12**	**2.28**	2.58
	Smoking activity				
ERD [67]	2.34	2.74	3.73	3.68	3.82
LSTM-3LR	2.05	2.34	3.10	3.24	3.42
S-RNN	**1.90**	**2.30**	**2.90**	**3.21**	**3.23**
	Discussion activity				
ERD [67]	2.67	2.97	3.23	3.47	2.92
LSTM-3LR	2.25	2.33	2.45	2.48	2.93
S-RNN	**1.67**	**2.03**	**2.20**	**2.39**	**2.43**

the ground truth in Table 5.1. Qualitatively, ERD models human motion better than LSTM-3LR. However, in the short-term, it does not mimic the ground-truth as well as LSTM-3LR. Fragkiadaki et al. [67] also note this trade-off between ERD and LSTM-3LR. On the other hand, S-RNN outperforms both LSTM-3LR and ERD on short-term motion forecasting on all activities. Therefore S-RNN gives us the best of both worlds. It mimics the ground truth in the short-term

and generates human like motion in the long term. Due to stochasticity in human motion, long-term forecasts (> 500ms) can significantly differ from the ground truth but still depict human-like motion. For this reason, the long-term forecast numbers in Table 5.1 are not a fair representative of algorithms modeling capabilities. We also observe that discussion is one of the most challenging aperiodic activity for all algorithms.

User study. We randomly sampled three seed motions from each of the four activities (walking, eating, smoking, and discussion), giving a total of 12 seed motions. We forecasted human motion from the seeds using S-RNN, LSTM-3LR and ERD, resulting in total of 36 forecasted motions – equally divided across algorithms and activities. We asked five users to rate the forecasted motions on a Likert scale of 1 – 3, where a score of 1 is bad, 2 is neutral, and 3 is good. The users were instructed to rate based on how human like the forecasted motion appeared. In order to calibrate, the users were first shown many examples of ground truth motion capture videos.

Figure 5.7 shows the number of examples that obtained bad, neutral, and good scores for each algorithm. Majority of the motions generated by S-RNN were of high-quality and resembled human like motion. On the other hand, LSTM-3LR generated reasonable motions most of the times, however they were not as good as the ground truth. Finally, the motions forecasted by ERD were not human like for most of the aperiodic activities (eating, smoking, and discussion). On the walking activity, all algorithms were competitive and users mostly gave a score of 3 (good). Hence, through the user study we validate that S-RNN generates most realistic human motions majority of the times.

To summarize, unstructured approaches like LSTM-3LR and ERD struggles to

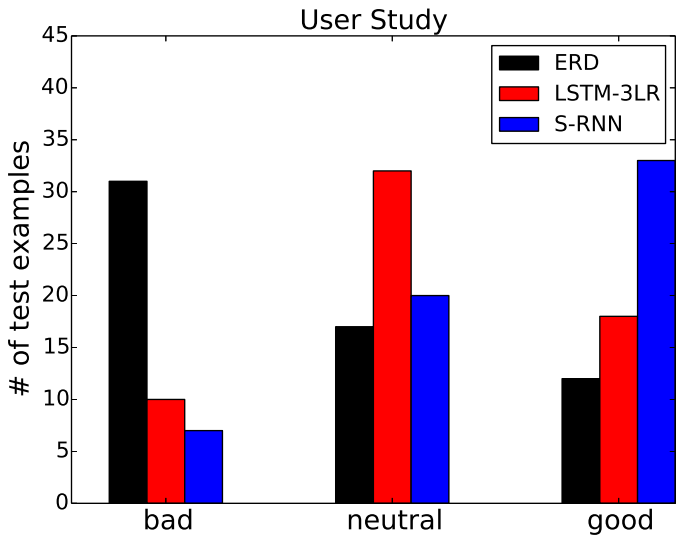


Figure 5.7: **User study** with five users. Each user was shown 36 forecasted motions equally divided across four activities (walking, eating, smoking, discussion) and three algorithms (S-RNN, ERD, LSTM-3LR). The plot shows the number of bad, neutral, and good motions forecasted by each algorithm.

model long-term human motion on complex activities. S-RNN's good performance is attributed to its structural modeling of human motion through the underlying st-graph. S-RNN models each body part separately with nodeRNNs and captures interactions between them with edgeRNNs in order to produce coherent motions.

5.4.2 Going deeper into structural-RNN

We now present several insights into S-RNN architecture and demonstrate the modularity of the architecture which enables it to generate hybrid human motions.

Visualization of memory cells. We investigated if S-RNN memory cells repre-

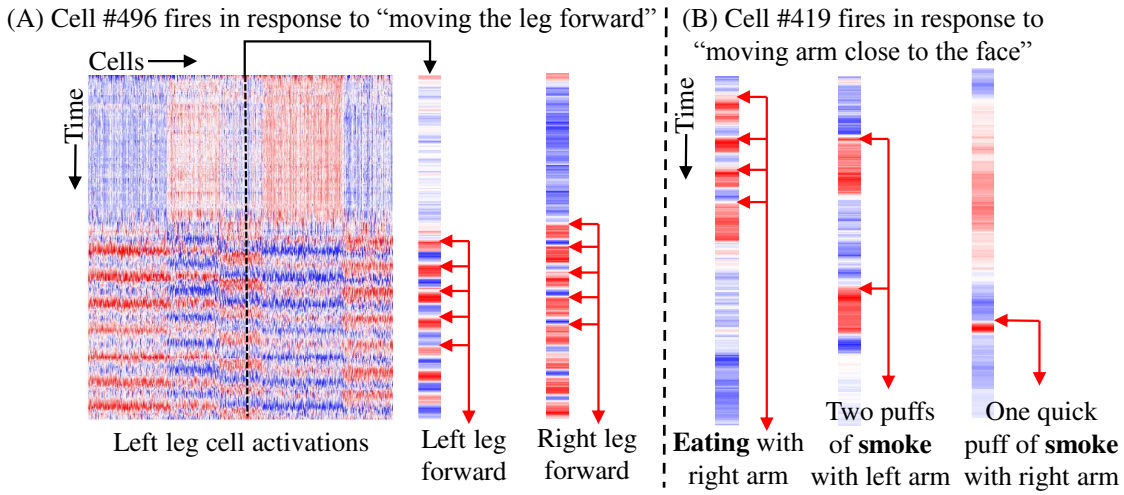


Figure 5.8: **S-RNN memory cell visualization.** (**Left**) A cell of the leg nodeRNN fires (red) when "putting the leg forward". (**Right**) A cell of the arm nodeRNN fires for "moving the hand close to the face". We visualize the same cell for eating and smoking activities. (See video)

sent meaningful semantic sub-motions. Semantic cells were earlier studied on other problems [109], we are the first to present it for human motion. In Figure 5.8 (left) we show a cell in the leg nodeRNN learns the semantic motion of *moving the leg forward*. The cell fires positive (red color) on the forward movement of the leg and negative (blue color) on its backward movement. As the subject walks, the cell alternatively fires for the right and the left leg. Longer activations in the right leg corresponds to the longer steps taken by the subject with the right leg. Similarly, a cell in the arm nodeRNN learns the concept of *moving hand close to the face*, as shown in Figure 5.8 (right). The same cell fires whenever the subject moves the hand closer to the face during eating or smoking. The cell remains active as long as the hand stays close to the face.

Generating hybrid human motion. We now demonstrate the flexibility of our modular architecture by generating novel yet meaningful motions which are

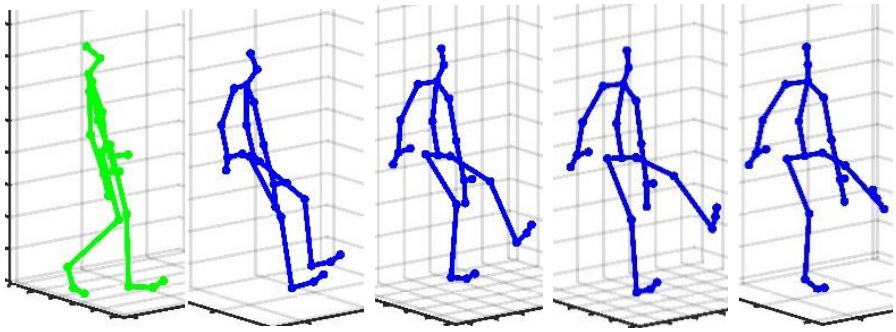


Figure 5.9: **Generating hybrid motions**. We demonstrate flexibility of S-RNN by generating a hybrid motion of a "human jumping forward on one leg". See video: `http://asheshjain.org/srnn`

not in the data set. Such modularity is of interest and has been explored to generate diverse motion styles [219]. As a result of having an underlying high-level structure, our approach allows us to exchange RNNs between the S-RNN architectures trained on different motion styles. We leverage this to create a novel S-RNN architecture which generates a hybrid motion of a *human jumping forward on one leg*, as shown in Figure 5.9. For this experiment we modeled the left and right leg with different nodeRNNs. We trained two independent S-RNN models – a slower human and a faster human (by down sampling data) – and swapped the left leg nodeRNN of the trained models. The resulting faster human, with a slower left leg, jumps forward on the left leg to keep up with its twice faster right leg.[4] Unstructured architectures like ERD [67] does not offer this kind of flexibility.

Visualization of train and test errors. Figure 5.10 examines the test and train error with iterations. Both S-RNN and ERD converge to similar training error, however S-RNN generalizes better with a smaller test error for next step prediction. The number of parameters in S-RNN are marginally more than ERD.

[4]Imagine your motion forward if someone holds your right leg and run!

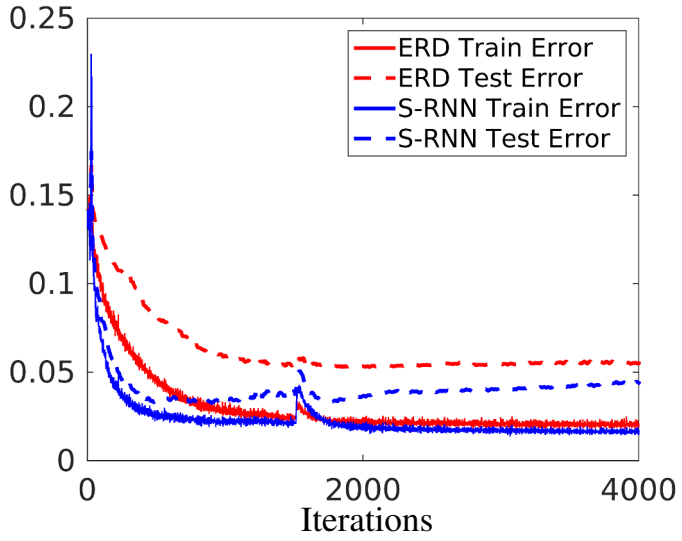


Figure 5.10: **Train and test error**. S-RNN generalizes better than ERD with a smaller test error.

S-RNN have more LSTMs than ERD, but each LSTM in S-RNN is half the size of the LSTMs used in ERD. For ERD we used the best set of parameters described in [67]. There, the authors cross-validated over model parameters. In the plot, the jump in error around iteration 1500 corresponds to the decay in step size. Due to addition of noise the test error of S-RNN exhibits a small positive slope, but it always stays below ERD.

5.4.3 Human activity detection and anticipation

In this section we present S-RNN for modeling human activities. We consider the CAD-120 [119] data set where the activities involve rich human-object interactions. Each activity consist of a sequence of sub-activities (e.g. moving, drinking etc.) and objects affordance (e.g., reachable, drinkable etc.), which evolves as the activity progresses. Detecting and anticipating the sub-activities and affordance enables personal robots to assist humans. However, the problem is

challenging as it involves complex interactions – humans interact with multiple objects during an activity, and objects also interact with each other (e.g. pouring water from "glass" into a "container"), which makes it a particularly good fit for evaluating our method. Koppula et al. [121, 119] represents such rich spatio-temporal interactions with the st-graph shown in Figure 5.5b, and models it with a spatio-temporal CRF. In this experiment, we show that modeling the same st-graph with S-RNN yields superior results. We use the node and edges features from [119].

Figure 5.3b shows our S-RNN architecture to model the st-graph. Since the number of objects varies with environment, factor sharing between the object nodes and the human-object edges becomes crucial. In S-RNN, $\mathbf{R}_{V_2}$ and $\mathbf{R}_{E_1}$ handles all the object nodes and the human-object edges respectively. This allows our fixed S-RNN architecture to handle varying size st-graphs. For edgeRNNs we use a single layer LSTM of size 128, and for nodeRNNs we use LSTM(256)-softmax(·). At each time step, the human nodeRNN outputs the sub-activity label (10 classes), and the object nodeRNN outputs the affordance (12 classes). Having observed the st-graph upto time t, the goal is to *detect* the sub-activity and affordance labels at the current time t, and also *anticipate* their future labels of the time step $t + 1$. For detection we train S-RNN on the labels of the current time step. For anticipation we train the architecture to predict the labels of the next time step, given the observations upto the current time. We also train a *multi-task* version of S-RNN, where we add two softmax layers to each nodeRNN and jointly train for anticipation and detection.

Table 5.2 shows the detection and anticipation F1-scores averaged over all the classes. S-RNN significantly improves over Koppula et al. on both anticipa-

Table 5.2: **Results on CAD-120 [119]**. S-RNN architecture derived from the st-graph in Figure 5.5b outperforms Koppula et al. [121, 119] which models the same st-graph in a probabilistic framework. S-RNN in multi-task setting (joint detection and anticipation) further improves the performance.

	Detection F1-score		Anticipation F1-score	
Method	Sub-activity (%)	Object Affordance (%)	Sub-activity (%)	Object Affordance (%)
Koppula et al. [121, 119]	80.4	81.5	37.9	36.7
S-RNN w/o edgeRNN	82.2	82.1	64.8	72.4
S-RNN	**83.2**	88.7	62.3	80.7
S-RNN (multi-task)	82.4	**91.1**	**65.6**	**80.9**

tion [121] and detection [119]. On anticipating object affordance S-RNN F1-score is 44% more than [121], and 7% more on detection. S-RNN does not have any Markov assumptions like spatio-temporal CRF, and therefore, it better models the long-time dependencies needed for anticipation. The table also shows the importance of edgeRNNs in handling spatio-temporal components. EdgeRNN transfers the information from the human to objects, which helps is predicting the object labels. Therefore, S-RNN without the edgeRNNs poorly models the objects. This signifies the importance of edgeRNNs and also validates our design. Finally, training S-RNN in a multi-task manner works best in majority of the cases. In Figure 5.11 we show the visualization of an eating activity. We show one representative frame from each sub-activity and our corresponding predictions.

S-RNN complexity. In terms of complexity, we discuss two aspects as a function of the underlying st-graph: (i) the number of RNNs in the mixture; and (ii) the complexity of forward-pass. The number of RNNs depends on the number

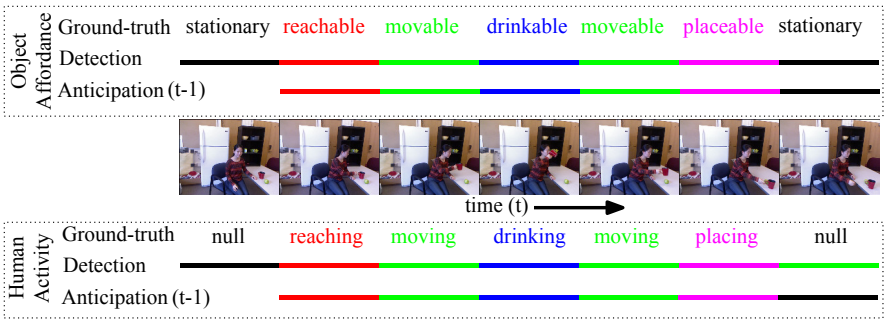


Figure 5.11: **Qualitative result on eating activity on CAD-120.** Shows multi-task S-RNN detection and anticipation results. For the sub-activity at time t, the labels are anticipated at time $t-1$.

of semantically similar nodes in the st-graph. The overall S-RNN architecture is compact because the edgeRNNs are shared between the nodeRNNs, and the number of semantic categories are usually few in context-rich applications. Furthermore, because of factor sharing the number of RNNs does not increase if more semantically similar nodes are added to the st-graph. The forward-pass complexity depends on the number of RNNs. Since the forward-pass through all edgeRNNs and nodeRNNs can happen in parallel, in practice, the complexity only depends on the cascade of two neural networks (edgeRNN followed by nodeRNN).

5.4.4 Driver maneuver anticipation

We now present S-RNN for another application which involves anticipating maneuvers several seconds before they happen. For example, anticipating a future lane change maneuver several seconds before the wheel touches the lane markings. This problem requires spatial and temporal reasoning of the driver, and the sensory observations from inside and outside of the car. It can be repre-

sented with the st-graph shown in Figure 5.5c. The st-graph represents the interactions between the observations outside the vehicle (eg. the road features), the driver's maneuvers, and the observations inside the vehicle (eg. the driver's facial features). In Chapter 4 (Section 4.8) we proposed a Bayesian network AIO-HMM for modeling this problem. AIO-HMM is a probabilistic approach for modeling the st-graph in Figure 5.5c. We now model the same st-graph with S-RNN architecture using the features we described in Chapter 4.

The nodeRNN models the driver, and the two edgeRNNs model the interactions between the driver and the observations inside the vehicle, and the observations outside the vehicle. The driver node is labeled with the future maneuver and, the observation nodes do not carry any label. The output of the driver nodeRNN is softmax probabilities of the following five maneuvers: {*Left lane change, right lane change, left turn, right turn, straight driving*}. Our nodeRNN architecture is FC(64)-softmax(5), and edgeRNN is LSTM(64). The resulting S-RNN architecture is same as the Fusion-RNN architecture we proposed in Section 4.6.2. Therefore, the Fusion-RNN and AIO-HMM proposed in Chapter 4 are alternate approaches for modeling the same st-graph. In Chapter 4 we also showed that Fusion-RNN outperforms AIO-HMM by a significant margin.

5.5 Conclusion

In this chapter we proposed a generic and principled approach for combining high-level spatio-temporal graphs with sequence modeling success of RNNs. We make use of factor graph, and factor sharing in order to obtain an RNN mixture that is scalable and applicable to any problem expressed over st-graphs.

Our RNN mixture captures the rich interactions in the underlying st-graph through connections between the RNNs. It learns the dependencies between the output labels by sharing RNNs between the outputs.

We demonstrated significant improvements with S-RNN on three diverse spatio-temporal problems. We showed that representing human motion over st-graph and learning via S-RNN outperforms state-of-the-art RNN based methods. We further showed, on two context-rich spatio-temporal problems: (i) human-object interaction; and (ii) driver maneuver anticipation; that learning S-RNN from their st-graphs outperforms the existing state-of-the-art non-deep learning based methods on the same st-graph. By visualizing the memory cells we showed that S-RNN learns certain semantic sub-motions, and demonstrated its modularity by generating hybrid human motions. Future work includes combining S-RNN with CNNs for spatio-temporal feature learning [229], and developing inference methods on S-RNN for structured-output prediction.

CHAPTER 6

KNOWLEDGE-ENGINE FOR ROBOTS

Typically every robot learn in isolation from other robots. While many research groups teach robots different concepts about the physical world such as grasping, affordances etc., there does not exist a common platform for robots to exchange and share the learned knowledge. In this chapter we introduce a knowledge-engine, which learns and shares knowledge representations, for robots to carry out a variety of tasks. Building such an engine brings with it the challenge of dealing with multiple data modalities including symbols, natural language, haptic senses, robot trajectories, visual features and many others. The *knowledge* stored in the engine comes from multiple sources including physical interactions that robots have while performing tasks (perception, planning and control), knowledge bases from the Internet and learned representations from several robotics research groups.

Contributions of this chapter. This knowledge-engine is developed in collaboration with several co-authors (Saxena, Jain, Sener, Jami, Misra, and Koppula [194]), and we call it RoboBrain. This dissertation primarily contributes the overall system architecture of RoboBrain, and its application in planning good trajectories. For completeness, in this chapter we also describe the formal definition of knowledge-engine, and the query language for accessing the knowledge. RoboBrain is open-source and available at: `http://www.robobrain.me`

6.1 Why do robots need a knowledge engine?

Over the last decade, we have seen many successful applications of large-scale knowledge systems. Examples include Google knowledge graph [50], IBM Watson [63], Wikipedia, and many others. These systems know answers to many of our day-to-day questions, and not crafted for a specific task, which makes them valuable for humans. Inspired by them, researchers have aggregated domain specific knowledge by mining data [8, 23], and processing natural language [30], images [46] and speech [160]. These sources of knowledge are specifically designed for humans, and their human centric design makes them of limited use for robots—for example, imagine a robot querying a search engine for how to "bring sweet tea from the kitchen" (Figure 6.1).

In order to perform a task, robots require access to a large variety of information with finer details for performing perception, planning, control and natural language understanding. When asked to bring sweet tea, as shown in Figure 6.1, the robot would need access to the knowledge for grounding the language symbols into physical entities, the knowledge that sweet tea can either be on a table or in a fridge, and the knowledge for inferring the appropriate plans for grasping and manipulating objects. Efficiently handling this joint knowledge representation across different tasks and modalities is still an open problem.

In this chapter we present RoboBrain that allows robots to learn and share such representations of knowledge. We learn these knowledge representations from a variety of sources, including interactions that robots have while performing perception, planning and control, as well as natural language and vi-

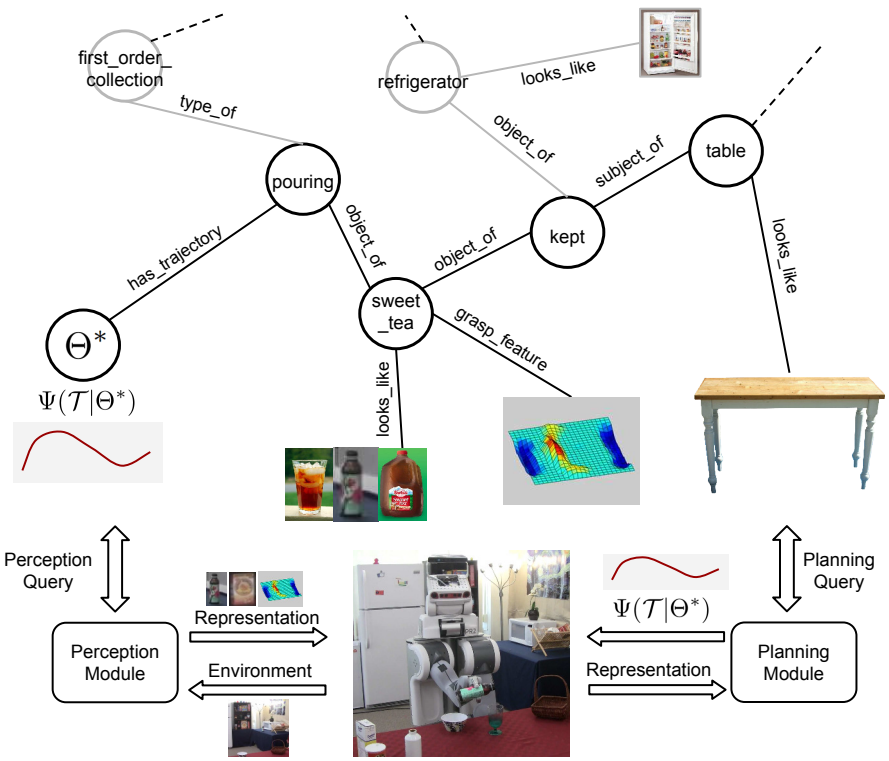


Figure 6.1: **An example showing a robot using RoboBrain for performing tasks.** The robot is asked "Bring me sweet tea from the kitchen", where it needs to translate the instruction into the perceived state of the environment. RoboBrain provides useful knowledge to the robot for performing the task: (a) sweet tea can be kept on a table or inside a refrigerator, (b) bottle can be grasped in certain ways, (c) opened sweet tea bottle needs to be kept upright, (d) the pouring trajectory should obey user preferences of moving slowly to pour, and so on.

sual data from the Internet. Our representation considers several modalities including symbols, natural language, visual or shape features, haptic properties, and so on. RoboBrain connects this knowledge from various sources and allow robots to perform diverse tasks by jointly reasoning over multiple data modalities.

RoboBrain enables sharing from multiple sources by representing the knowledge in a graph structure. Traversals on the RoboBrain graph allow robots to

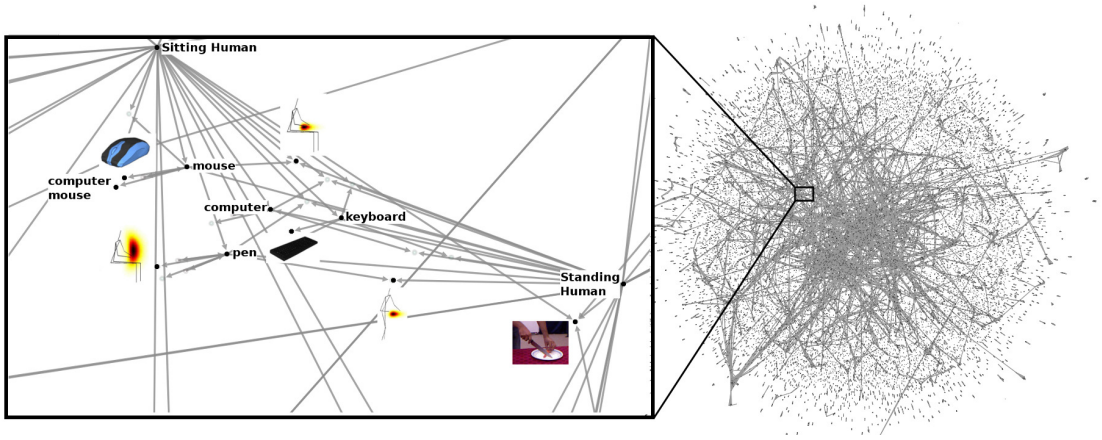


Figure 6.2: **A visualization of the RoboBrain graph** on Nov 2014, showing about 45K nodes and 100K directed edges. The left inset shows a zoomed-in view of a small region of the graph with rendered media. This illustrates the relations between multiple modalities namely images, heatmaps, words and human poses. For high-definition graph visualization, see: `http://pr.cs.cornell.edu/robobrain/graph.pdf`

gather the specific information they need for a task. This includes the semantic information, such as different grasps of the same object, as well as the functional knowledge, such as spatial constraints (e.g., a bottle is kept on the table and not the other way around). The key challenge lies in building this graph from a variety of knowledge sources while ensuring dense connectivity across nodes. Furthermore, there are several challenges in building a system that allows concurrent and distributed update, and retrieval operations.

We present use of RoboBrain *as-a-service*, which allow researchers to effortlessly use the state-of-the-art algorithms. We also present experiments to show that sharing knowledge representations through RoboBrain improves existing path planning algorithms. RoboBrain is a collaborative project that we support by designing a large-scale cloud architecture. In the current state, RoboBrain stores and shares knowledge across several research projects [223, 159, 96, 99,

100, 121, 246, 139] and Internet knowledge sources [138, 61]. RoboBrain knowledge graph is available at `http://www.robobrain.me`.

In this chapter we summarize the key ideas and challenges in building a knowledge-engine for robots. The goal of this chapter is to present an overall view of the RoboBrain, its architecture, functionalities, and demonstrate its application to robotics. In Section 6.4 we formally define the RoboBrain graph and describe its system architecture in Section 6.5. In order for robots to use RoboBrain we propose the Robot Query Library in Section 6.6. In Section 6.7 we present the application of RoboBrain in path planning.

6.2 Related work

We now describe some works related to RoboBrain. We first give an overview of the existing knowledge bases and describe how RoboBrain differs from them. We then describe some works in robotics that can benefit from RoboBrain, and also discuss some of the related on-going efforts.

Knowledge bases. Collecting and representing a large amount of information in a knowledge base (KB) has been widely studied in the areas of data mining, natural language processing and machine learning. Early seminal works have manually created KBs for the study of common sense knowledge (Cyc [138]) and lexical knowledge (WordNet [61]). With the growth of Wikipedia, KBs started to use crowdsourcing (DBPedia [8], Freebase [23]) and automatic information extraction (Yago [210, 85], NELL [30]) for mining knowledge.

One of the limitations of these KBs is their strong dependence on a single

modality that is the text modality. There have been few successful attempts to combine multiple modalities. ImageNet [46] and NEIL [34] enriched text with images obtained from Internet search. They used crowdsourcing and unsupervised learning to get the object labels.

We have seen successful applications of the existing KBs within the modalities they covered, such as IBM Watson Jeopardy Challenge [64]. However, the existing KBs are human centric and do not directly apply to robotics. The robots need finer details about the physical world, e.g., how to manipulate objects, how to move in an environment, etc. In RoboBrain we combine knowledge from the Internet sources with finer details about the physical world, from RoboBrain project partners, to get an overall rich graph representation.

Robot Learning. For robots to operate autonomously they should perceive their environments, plan paths, manipulate objects and interact with humans. We describe previous work in each of these areas and how RoboBrain complements them.

Perceiving the environment. Perception is a key element of many robotic tasks. It has been applied to object labeling [134, 4, 246], scene understanding [113, 80], robot localization [155, 165], path planning [112], and object affordances [45, 125]. RoboBrain stores perception related knowledge in the form of 3D point clouds, grasping features, images and videos. It also connects this knowledge to human understandable concepts from the Internet knowledge sources.

Path planning and manipulation. Planning algorithms formulate action plans which are used by robots to move around and modify its environment. Planning algorithms have been proposed for the problems of motion planning [256, 195],

task planning [24] and symbolic planning [187]. Some planning applications include robots baking cookies [24], folding towels [151], assembling furniture [115], and preparing pancakes [11]. The previous works have also learned planning parameters using Inverse Optimal Control [181, 96]. RoboBrain stores the planning parameters learned by previous works and allow the robots to query for the parameters.

Interacting with humans. Human-robot interaction includes collaborative tasks between humans and robots [168], generating safe and human-like robot motion [149, 71, 55, 28], interaction through natural language [222, 158], etc. These applications require joint treatment of perception, manipulation and natural language understanding. RoboBrain stores different data modalities required by these applications.

Previous efforts on connecting robots range from creating a common operating system (ROS) for robots [178] to sharing data acquired by various robots via cloud [239, 5]. For example, the RoboEarth [239] provides a platform for the robots to store and off-load computation to the cloud and communicate with other robots; and the KIVA systems [5] use the cloud to coordinate motion for hundreds of mobile platforms. On the other hand, RoboBrain provides a knowledge representation layer on top of data storing, sharing and communication.

Open-Ease [12] is a related on-going effort towards building a knowledge-engine for robots. Open-Ease and RoboBrain differ in the way they learn and represent knowledge. In Open-Ease the knowledge is represented as formal statements using pre-defined templates. On the other hand, the knowledge in RoboBrain is represented as a graph. The nodes of the RoboBrain graph have no pre-defined templates and they can be any robotic concept like grasping fea-

tures, trajectory parameters, and visual data. This graph representation allows partner projects to easily integrate their learned concepts in RoboBrain. The semantic meaning of concepts in the RoboBrain graph are represented by their connectivity patterns in the graph.

6.3 Overview

RoboBrain is a never ending learning system that continuously incorporates new knowledge from its partner projects and from different Internet sources. One of the functions of RoboBrain is to represent the knowledge from various sources as a graph, as shown in Figure 6.2. The nodes of the graph represent concepts and edges represent the relations between them. The connectivity of the graph is increased through a set of graph operations that allow additions, deletions and updates to the graph. So far RoboBrain has successfully connected knowledge from sources like WordNet, ImageNet, Freebase, OpenCyc, parts of Wikipedia and other partner projects. These knowledge sources provide lexical knowledge, grounding of concepts into images and common sense facts about the world.

The knowledge from the partner projects and Internet sources can sometimes be erroneous. RoboBrain handles inaccuracies in knowledge by maintaining beliefs over the correctness of the concepts and relations. These beliefs depend on how much RoboBrain trusts a given source of knowledge, and also the feedback it receives from crowdsourcing (described below). For every incoming knowledge, RoboBrain also makes a sequence of decisions on whether to form new nodes, or edges, or both. Since the knowledge carries semantic meaning

RoboBrain makes many of these decisions based on the contextual information that it gathers from nearby nodes and edges. For example, RoboBrain resolves polysemy using the context associated with nodes. Resolving polysemy is important because a 'plant' could mean a 'tree' or an 'industrial plant' and merging the nodes together will create errors in the graph.

RoboBrain incorporates supervisory signals from humans in the form of crowdsourcing feedback. This feedback allows RoboBrain to update its beliefs over the correctness of the knowledge, and to modify the graph structure if required. While crowdsourcing feedback was used in some previous works as means for data collection (e.g., [46, 191]), in RoboBrain they serve as supervisory signals that improve the knowledge-engine. RoboBrain allows user interactions at multiple levels: (i) Coarse feedback: these are binary feedback where a user can "Approve" or "Disapprove" a concept in RoboBrain through its online web interface; (ii) Graph feedback: these feedback are elicited on RoboBrain *graph visualizer*, where a user modifies the graph by adding/deleting nodes or edges; (iii) Robot feedback: these are the physical feedback given by users directly on the robot.

In this chapter we discuss different aspects of RoboBrain, and show how RoboBrain serves as a knowledge layer for the robots. In order to support knowledge sharing, learning, and crowdsourcing feedback we develop a large-scale distributed system. We describe the architecture of our system in Section 6.5. In Section 6.6 we describe the robot query library, which allow robots to interact with RoboBrain. Through experiments we show that robots can use RoboBrain *as-a-service* and that knowledge sharing through RoboBrain improves existing robotic applications such as path planning.

6.4 Knowledge engine formal definition

In this section we present the formal definition of RoboBrain. RoboBrain represents knowledge as a directed graph $\mathcal{G} = (V, E)$. The vertices V of the graph stores concepts that can be of a variety of types such as images, text, videos, haptic data, or learned entities such as affordances, deep learning features, parameters, etc. The edges $E \subseteq V \times V \times C$ are directed and represents the relations between concepts. Each edge has an edge-type from a set C of possible edge-types.

An edge (v_1, v_2, ℓ) is an ordered set of two nodes v_1 and v_2 and an edge-type ℓ. Few examples of such edges are: (StandingHuman, Shoe, *CanUse*), (StandingHuman, $\mathcal{N}(\mu, \Sigma)$, *SpatiallyDistributedAs*) and (Grasping, DeepFeature23, *UsesFeature*). We do not impose any constraints on the type of data that nodes can represent. However, we require the edges to be consistent with RoboBrain edge set C. We further associate each node and edge in the graph with *a feature vector representation* and a *belief*. The feature vector representation of nodes and edges depend on their local connections in the graph, and their belief is a scalar probability over the accuracy of the information that the node or an edge represents. Tables 6.1 and 6.2 show few examples of nodes and edge-types. A snapshot of the graph is shown in Figure 6.2.

Creating the Graph. Graph creation consists of never ending cycle of two stages namely, knowledge acquisition and inference. Within the knowledge acquisition stage, we collect data from various sources and during the inference stage we apply statistical techniques to update the graph structure based on the aggregated data. We explain these two stages below.

Word	an English word represented as an ASCII string
DeepFeature	feature function trained with a Deep Neural Network
Image	2D RGB Image
PointCloud	3D point cloud
Heatmap	heatmap parameter vector

Table 6.1: Some examples of different node types in our RoboBrain graph. For full-list, please see the code documentation.

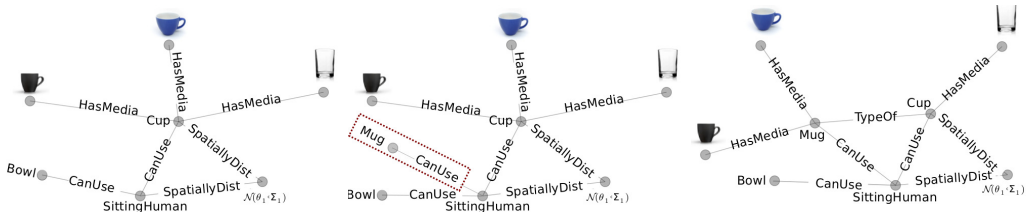
IsTypeOf	human *IsTypeOf* a mammal
HasAppearance	floor *HasAppearance* as follows (this image)
CanPerformAction	human *CanPerformAction* cutting
SpatiallyDistributedAs	location of human is *SpatiallyDistributedAs*
IsHolonym	tree *IsHolonym* of leaf

Table 6.2: Some examples of different edge types in our RoboBrain graph. For full-list, please see the code documentation.

1. Knowledge acquisition: RoboBrain accepts new information in the form of set of edges, which we call a *feed*. A *feed* can either be from an automated algorithm crawling the Internet sources or from one of RoboBrain's partner projects. We add a new *feed* to the existing graph through a sequence of union operations performed on the graph. These union operations are then followed by an inference algorithm. More specifically, given a new *feed* consisting of a set of N edges $\{(v_1^1, v_2^1, \ell^1) \ldots (v_1^N, v_2^N, \ell^N)\}$, and the existing graph $G = (V, E)$. The graph union operations give a graph $G' = (V', E')$ as follows:

$$\begin{aligned} V' &= v_1^1 \cup v_2^1 \cup \ldots \cup v_1^N \cup v_2^N \cup V \\ E' &= (v_1^1, v_2^1, \ell^1) \cup \ldots \cup (v_1^N, v_2^N, \ell^N) \cup E \end{aligned} \tag{6.1}$$

2. Inference on the Graph: After adding the *feed* to the graph using equation (6.1), we update the graph based on this new knowledge. This step outputs a se-



(a) Original graph (b) Feed insertion (c) after $merge(Mug, Mug')$ $\rightarrow$ $Mug \circ split(Cup)$ $\rightarrow$ (Cup, Mug')

Figure 6.3: **Visualization of inserting new information.** We insert '*Sitting human can use a mug*' and RoboBrain infers the necessary split and merge operations on the graph. In (a) we show the original sub-graph, In (b) information about a *Mug* is seen for the first time and the corresponding node and edge are inserted, In (c) inference algorithm infers that previously connected cup node and cup images are not valid any more, and it splits the *Cup* node into two nodes as *Cup* and *Mug*′ and then merges *Mug*′ and *Mug* nodes.

quence of graph operations which are then performed on the graph. These graph operations modify the graph by adding new nodes or edges to the graph, deleting nodes or edges from the graph, merging or splitting nodes, etc.

We mention two graph operations here: *split* and *merge*. The split operation is defined as splitting a node into a set of two nodes. The edges having end points in the split node are connected to one of the resultant nodes. A merge operation is defined as merging two nodes into a single node, while updating the edges connected to the merged nodes. An example of such an update is shown in Figure 6.3. When a new information *"sitting human can use a mug"* is added to the graph, it causes the *split* of the *Cup* node into two nodes: a *Cup* and a *Mug* node. These two are then connected by an edge-type *TypeOf*. Thorough description of the inference steps is beyond the scope of this chapter, for more details see Saxena et al. [194]. We now describe the system architecture of RoboBrain.

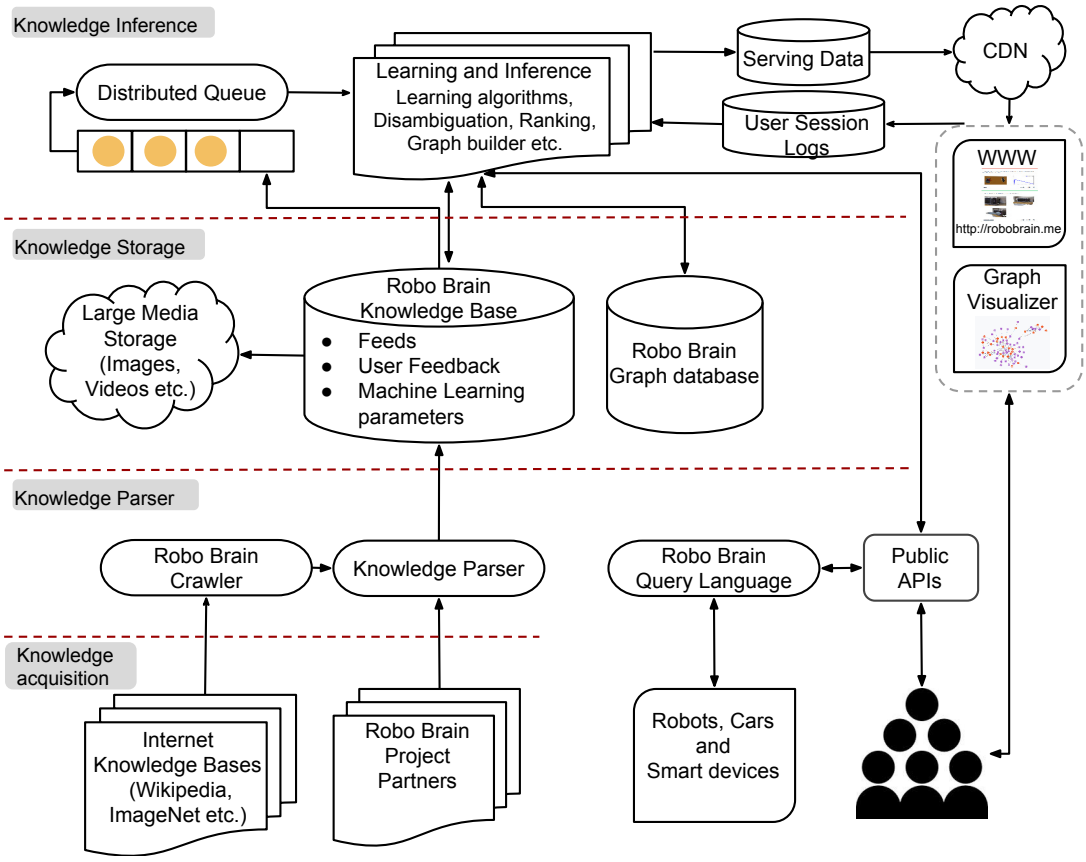


Figure 6.4: **RoboBrain system architecture.** It consists of four interconnected knowledge layers and supports various mechanisms for users and robots to interact with RoboBrain.

6.5 System architecture

We now describe the system architecture of RoboBrain, shown in Figure 6.4. The system consists of four interconnected layers: (a) knowledge acquisition, (b) knowledge parser, (c) knowledge storage, and (d) knowledge inference. The principle behind our design is to efficiently process large amount of unstructured multi-modal knowledge and represent it using the structured RoboBrain graph. In addition, our design also supports various mechanisms for users and robots to interact with RoboBrain. Below we discuss each of the components.

Knowledge acquisition layer is the interface between RoboBrain and different sources of multi-modal data. Through this layer RoboBrain gets access to new information which the other layers process. RoboBrain primarily collects knowledge through partner projects, by crawling existing knowledge bases such as Freebase, ImageNet, WordNet, etc., and from unstructured sources such as Wikipedia.

Knowledge parser layer of RoboBrain processes the data acquired by the acquisition layer and converts it to a consistent format for the storage layer. It also marks the incoming data with appropriate meta- data such as timestamps, source version number etc., for scheduling and managing future data processing. Moreover, since the knowledge bases might change with time, it adds a back pointer to the original source.

Knowledge storage layer of RoboBrain is responsible for storing different representations of the data. In particular it consists of a NoSQL document storage database cluster – RoboBrain Knowledge Base (RoboBrain-KB) – to store "feeds" parsed by the knowledge parser, crowdsourcing feedback from users, and parameters of different machine learning algorithms provided by RoboBrain project partners. RoboBrain-KB offloads large media content such as images, videos and 3D point clouds to a distributed object storage system built using Amazon Simple Storage Service (S3). The real power of RoboBrain comes through its graph database (RoboBrain-GD) which stores the structured knowledge. The data from RoboBrain-KB is refined through multiple learning algorithms and its graph representation is stored in RoboBrain-GD. The purpose behind this design is to keep RoboBrain-KB as the RoboBrain's single source of truth (SSOT). SSOT centric design allows us to re-build RoboBrain-GD in case

of failures or malicious knowledge sources.

Knowledge inference layer contains the key processing and machine learning components of RoboBrain. New and recently updated feeds go through a persistent replicated distributed queuing system (Amazon SQS), which are then consumed by some of our machine learning plugins (inference algorithm, graph builder, etc.) and populates the graph database. These plugins along with other learning algorithms (operating on the entire graph) constitute our learning and inference framework.

RoboBrain supports various *interaction mechanisms* to enable robots and users to communicate with the knowledge-engine. We develop a Robot Query Library as a primary method for robots to interact with RoboBrain. We also make available a set of public APIs to allow information to be presented on the World Wide Web (WWW) for online learning mechanisms (eg., crowdsourcing). RoboBrain serves all its data using a commercial content delivery network (CDN) to reduce the end user latency.

6.6 Robot Query Library (RQL)

In this section we present the RQL query language, through which the robots use RoboBrain for various robotic applications. The RQL provides a rich set of *retrieval functions* and *programming constructs* to perform complex traversals on the RoboBrain graph. An example of such a query is finding the possible ways for humans to use a cup. This query requires traversing paths from the human node to the cup node in the RoboBrain graph.

RQL allows expressing both the *pattern* of sub-graphs to match and the *operations* to perform on the retrieved information. An example of such an operation is *ranking* the paths from the human to the cup node in the order of relevance. The RQL admits following two types of functions: (i) graph retrieval functions; and (ii) programming construct functions.

Graph retrieval function. The graph retrieval function is used to find sub-graphs matching a given *template* of the form: Template: $(u) \rightarrow [e] \rightarrow (v)$

In the template above, the variables u and v are nodes in the graph and the variable e is a directed edge from u to v. We represent the graph retrieval function with the keyword `fetch` and the corresponding RQL query takes the form: `fetch`(Template) This RQL query finds the sub-graphs matching the template. It instantiates the variables in the template to match the sub-graph and returns the list of instantiated variables. We now give a few use cases of the retrieval function for RoboBrain.

Example 6.6.1. RQL query to retrieve all the objects that a human can use

$$\text{Query: } \texttt{fetch}((\{\texttt{name} : \text{`}\texttt{Human}\text{'}\}) \rightarrow [\text{`}\texttt{CanUse}\text{'}] \rightarrow (\texttt{v}))$$

The above query returns a list of nodes that are connected to the node with name `Human` and with an edge of type `CanUse`.

Programming construct functions. The programming construct functions serve to process the sub-graphs retrieved by the graph retrieval function `fetch`. In order to define these functions we make use of functional programming constructs like `map`, `filter` and `find`. We now explain the use of some of these constructs in RQL.

Example 6.6.2. RQL query to retrieve affordances of all the objects usable by a human.

```
objects := fetch({name : 'Human'}) → ['CanUse'] → (v)

affordances n := fetch({name : n}) → ['HasAffordance'] → (v)

map(λu → affordances u) objects
```

In this example, we illustrate the use of `map` construct. The `map` takes as input a function and a list, and then applies the function to every element of the list. More specifically, in the example above, the function `objects` retrieves the list of objects that the human can use. The `affordances` function takes as input an object and returns its affordances. In the last RQL query, the `map` applies the function `affordances` to the list returned by the function `objects`. We will use `fetch` and `map` operations in the next section for path planning. See [194] for more examples of RQL.

6.7 Applications to path planning

In this section we first show how RoboBrain can be used *as-a-service* by the robots for path planning. We then show how RoboBrain can help robotics projects by sharing knowledge within the projects and throughout the Internet.

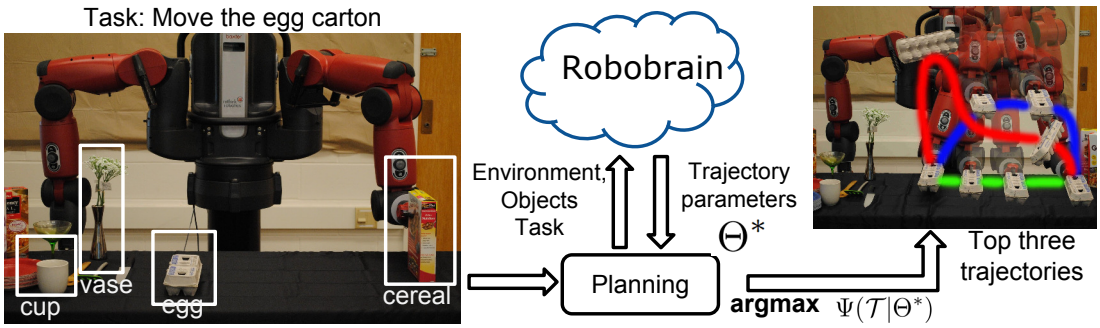


Figure 6.5: **RoboBrain for planning trajectory.** The robot queries RoboBrain for the trajectory parameters (learned by Jain et al. [96]) to plan paths for the fragile objects like an egg carton.

6.7.1 Use of knowledge engine *as-a-service*

Our goal with providing RoboBrain *as-a-service* is to allow robots to use the representations learned by different partner projects. This allows RoboBrain to effortlessly address many robotics applications. In the following we demonstrate RoboBrain as-a-service feature for path planning. In [194] we also demonstrate RoboBrain as-a-service feature for grounding natural language, and human-activity anticipation. However, their discussion is beyond the scope of this dissertation.

Path planning using RoboBrain

One key problem robots face in performing tasks in human environments is identifying trajectories desirable to the users. An appropriate trajectory not only needs to be geometrically valid (i.e., feasible and obstacle-free), but it also needs to satisfy the user preferences [96, 90]. For example, a robot should move sharp objects such as knife strictly away from nearby humans [94]. Such preferences

are commonly represented as cost functions which jointly model the environment, the task, and trajectories. Typically research groups have independently learned different cost functions [96, 130, 113], which are not shared across these groups. Here we show RoboBrain *as-a-service* for a robot to store and retrieve the planning parameters.

In Figure 6.5 we illustrate the robot planning for an egg carton by querying RoboBrain. Since eggs are *fragile*, users prefer to move them slowly and close to the surface of the table. In order to complete the task, the robot queries RoboBrain and retrieves the attributes of the egg carton and also the trajectory parameters learned in the previous work by Jain et al. [96]. Using the retrieved attributes and the parameters, the robot samples trajectories and executes the top-ranked trajectory. Below we show the RQL queries.

```
attributes n := fetch ({name : n}) → ['HasAttribute'] → (v)
trajectories a := fetch ({handle : a}) → ['HasTrajectory'] → (v)
trajectory_parameters := map(λa → trajectories a) attributes 'egg'
```

6.7.2 Improving path planning by knowledge sharing

RoboBrain allows sharing the knowledge learned by different research groups as well as knowledge obtained from various internet sources. In this section we show with an experiment how sharing knowledge improves path planning:

Sharing knowledge from the Internet. In this experiment we show that sharing knowledge from several Internet sources using RoboBrain improves robotic applications such as path planning. Knowledge from the Internet sources has

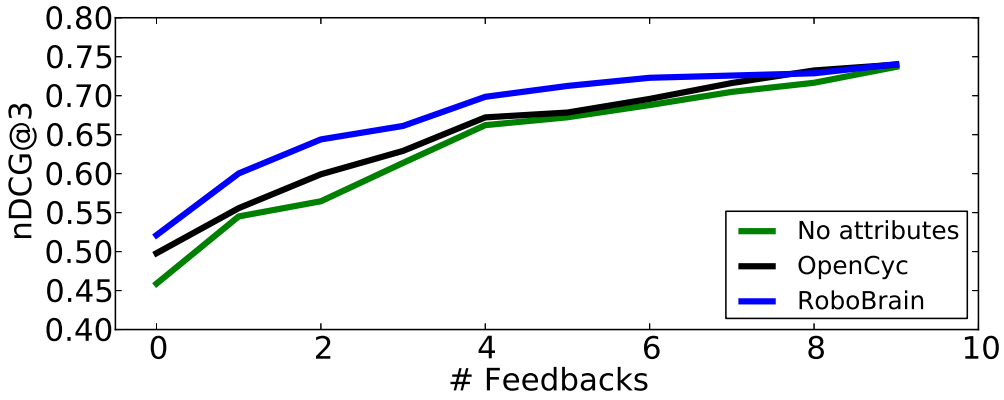


Figure 6.6: **Sharing from Internet sources.** The plot shows performance of the algorithm by Jain et al. [96] for three settings of attributes. This is an online algorithm that learns a good trajectory from the user feedback. The performance is measured using the nDCG metric [152], which represents the quality of the ranked list of trajectories. RoboBrain combines information from multiple sources and hence its richer in attributes as compared to retrieving attributes from OpenCyc alone.

been shown to help robots in planing better paths [224], understand natural language [189, 223], and also recently in object retrieval [78]. However, for certain robotic tasks a single Internet source does not cover many of the real world situations that the robot may encounter. In such situations it is desired to use multiple sources to get richer representation. The RoboBrain graph is designed to acquire and connect information from multiple Internet sources and make it accessible to robots.

In this experiment we build upon the work by Jain et al. [96] for planning trajectories that follow user preferences. The work relied on object attributes in order to plan desirable trajectories. These attributes convey properties such as whether an object is sharp, heavy, electronic etc. The attributes were manually defined by the authors [96]. In practice this is very challenging and time-

consuming because there are many objects and many attributes for each object. Instead of manually defining the attributes, we can retrieve many of them from the Internet knowledge sources such as OpenCyc, Wikipedia, etc. However, a single knowledge source might not have attributes for all objects. The RoboBrain graph connects many attributes obtained from multiple Internet sources to their respective objects.

Figure 6.6 illustrates the planning results when the robot does not use any attributes, when it uses attributes from a single source (OpenCyc), and when it use attributes from RoboBrain. The planning performance is best when using RoboBrain since it covers more attributes than the OpenCyc alone. Most importantly all these attributes are retrieved from the RoboBrain graph with a single RQL query as explained in Section 6.7.1.

6.8 Discussion and conclusion

RoboBrain graph currently has 44347 nodes (concepts) and 98465 edges (relations). The knowledge in the graph is obtained from the Internet sources and through the project partners. For the success of many robotics application it is important to relate and connect the concepts from these different knowledge sources. In Figure 6.7 we plot the degree distribution of the RoboBrain graph and compare it with the degree distribution of independent knowledge sources. The graph of independent knowledge sources is the union of each knowledge source, which have nodes from all the projects and the edges only between the nodes from the same project. RoboBrain successfully connects projects and increases the average degree per-node by 0.8. RoboBrain graph has fifteen thou-

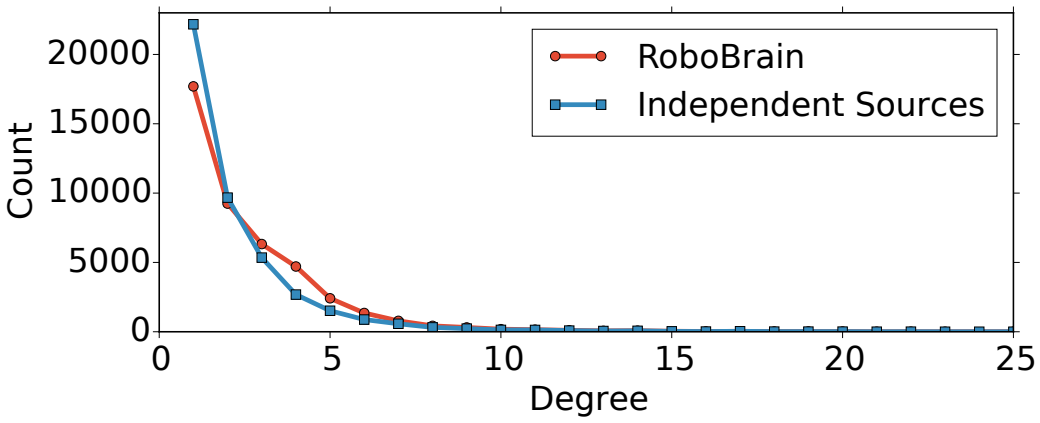


Figure 6.7: Degree distribution of RoboBrain and the union of independent knowledge sources. For the case of independent sources, we only consider the edges between nodes from the same source. RoboBrain connects different projects successfully: number of nodes with degree 1 and 2 decrease and nodes with degree 3 and more increase.

sand nodes with degree one. Most of these nodes come from Wikipedia and WordNet. These nodes are not directly related to the physical world and represent concepts like political ideas, art categories, etc.

In this chapter we described different aspects and technical challenges in building a knowledge-engine for robots. RoboBrain represents multiple data modalities from various sources, and connects them to get an overall rich graph representation. We presented an overview of the large-scale system architecture and developed the Robot Query Library (RQL) for robots to use RoboBrain. We illustrated path planning as simple RQL queries to RoboBrain. We also showed in experiments that sharing knowledge through RoboBrain improves existing path planning algorithms.

CHAPTER 7

CONCLUSION AND FUTURE WORK

While robots can learn certain skills from off-the-shelf available data sets. For many other skills they need to observe and interact with humans. Learning and improving from human interactions will be a critical component of the future robotic systems. Equally important will be the need to make the interactions natural to the (non-expert) end user. In this dissertation we proposed methods for learning from *natural* human interactions for several robotic problems. We focussed on interaction mechanisms that are scalable and easy to elicit from non-expert users. We applied our methods on robotic manipulators and assistive cars, and tested them in the real-world situations with many users.

We proposed a coactive learning framework for robotic manipulators to learn from non-expert users. In our framework the user iteratively improves the trajectory proposed by the robot but never reveals the optimal answer. In order to allow such iterative improvements, we proposed several interaction mechanisms that were intuitive to the end user. To scale the interactions further, we proposed a crowdsourcing system which allow the users to reveal their preferences by simply watching videos of robot navigating in human environments. Using our learning framework and the interaction mechanisms we trained robots to perform various household tasks in context rich environments.

Assistive cars, also a kind of robots, are growing in popularity because of their life saving features such as adaptive cruise control, automatic forward collision avoidance etc. Furthermore, driving a car is one of the most naturally occurring human-robot interaction. In this dissertation we addressed the question: *what a car can learn by observing its driver?* We proposed a vehicular sensory

platform which by simply observing many examples of driving learns to anticipate the maneuvers few seconds before they happen. Our learning algorithm consisits of a sensory fusion RNN architecture which simultaneously learns to anticipate and fuse the information from multiple sensors.

In order to address such spatio-temporal interactions more generally, we proposed a generic and principled approach for transforming the spatio-temporal graphs to structures of Recurrent Neural Networks (RNNs). Our approach allows domain experts to express their high-level reasoning's over spatio-temporal graphs and then learn an expressive deep architecture to capture their reasoning. We applied our framework on several problems, including to model human motion from motion capture data, for driving maneuver anticipation, and for understanding human-object interaction.

The future works could explore ways for extending learning from natural human interactions on both the application and algorithmic ends. We need methods for handling uncertainty in perception which our work currently does not address. This could require combining simultaneous localization and mapping (SLAM) methods with human-robot interactive learning. The field also needs a formal human-robot interaction study that compares different kinds of user feedback on their ease of use and the amount of information they carry. For rapid progress in this area we could also equip robots with special feedback interfaces that allow users to reveal their preferences with ease. Our current transformation of spatio-temporal graphs to RNNs does not handle structured output prediction [43], i.e. we predict each label independent of the other. This extension could also be an interesting direction for the community.

Reproducibility of human-robot interactive learning experiments in equally

important and needs to be addressed. This is also challenging since it involves the interplay between the robotic hardware, system integration, and the human subject. In order to make such experiments reproducible we should borrow ideas from the information retrieval community which has extensively published ways for evaluating search engines and their interaction with real-world users. Human-robot interactive learning systems should also take inspiration from open-source planning and perception libraries [178] that have greatly accelerated progress in their respective fields.

APPENDIX A

CHAPTER 2 APPENDIX

A.1 Proof for Average Regret

This proof builds upon Shivaswamy & Joachims [199].

We assume the user hidden score function $s^*(x, y)$ is contained in the family of scoring functions $s(x, y; w_O^*, w_E^*)$ for some unknown w_O^* and w_E^*. Average regret for TPP over T rounds of interactions can be written as:

$$\begin{aligned} REG_T &= \frac{1}{T}\sum_{t=1}^{T}(s^*(x_t, y_t^*) - s^*(x_t, y_t)) \\ &= \frac{1}{T}\sum_{t=1}^{T}(s(x_t, y_t^*; w_O^*, w_E^*) - s(x_t, y_t; w_O^*, w_E^*)) \end{aligned}$$

We further assume the feedback provided by the user is strictly α-informative and satisfy following inequality:

$$\begin{aligned} s(x_t, \bar{y}_t; w_O^*, w_E^*) \geq & s(x_t, y_t; w_O^*, w_E^*) + \alpha[s(x_t, y_t^*; w_O^*, w_E^*) \\ & - s(x_t, y_t; w_O^*, w_E^*)] - \xi_t \end{aligned} \tag{A.1}$$

Later we relax this constraint and requires it to hold only in expectation.

This definition states that the user feedback should have a score of $\bar{y}_t$ that is higher than that of y_t by a fraction $\alpha \in (0, 1]$ of the maximum possible range $s(x_t, y_t^*; w_O^*, w_E^*) - s(x_t, y_t; w_O^*, w_E^*)$.

Theorem 1. *The average regret of trajectory preference perceptron receiving strictly α-informative feedback can be upper bounded for any $[w_O^*; w_E^*]$ as follows:*

$$REG_T \leq \frac{2C \left\| [w_O^*; w_E^*] \right\|}{\alpha \sqrt{T}} + \frac{1}{\alpha T} \sum_{t=1}^{T} \xi_t \tag{A.2}$$

where C is constant such that $\left\| [\phi_O(x, y); \phi_E(x, y)] \right\|_2 \leq C$.

Proof: After T rounds of feedback, using weight update equations of w_E and w_O we can write:

$$w_O^* \cdot w_O^{(T+1)} = w_O^* \cdot w_O^{(T)} + w_O^* \cdot (\phi_O(x_T, \bar{y}_T) - \phi_O(x_T, y_T))$$
$$w_E^* \cdot w_E^{(T+1)} = w_E^* \cdot w_E^{(T)} + w_E^* \cdot (\phi_E(x_T, \bar{y}_T) - \phi_E(x_T, y_T))$$

Adding the two equations and recursively reducing the right gives:

$$w_O^* \cdot w_O^{(T+1)} + w_E^* \cdot w_E^{(T+1)} = \sum_{t=1}^{T} (s(x_t, \bar{y}_t; w_O^*, w_E^*) - s(x_t, y_t; w_O^*, w_E^*)) \tag{A.3}$$

Using Cauchy-Schwarz inequality the left hand side of equation (A.3) can be bounded as:

$$w_O^* \cdot w_O^{(T+1)} + w_E^* \cdot w_E^{(T+1)} \leq \left\| [w_O^*; w_E^*] \right\| \left\| [w_O^{(T+1)}; w_E^{(T+1)}] \right\| \tag{A.4}$$

$\left\|[w_O^{(T+1)}; w_E^{(T+1)}]\right\|$ can be bounded by using weight update equations:

$$
\begin{aligned}
& w_O^{(T+1)} \cdot w_O^{(T+1)} + w_E^{(T+1)} \cdot w_E^{(T+1)} = w_O^{(T)} \cdot w_O^{(T)} + w_E^{(T)} \cdot w_E^{(T)} \\
& + 2w_O^{(T)} \cdot (\phi_O(x_T, \bar{y}_T) - \phi_O(x_T, y_T)) \\
& + 2w_E^{(T)} \cdot (\phi_E(x_T, \bar{y}_T) - \phi_E(x_T, y_T)) \\
& + (\phi_O(x_T, \bar{y}_T) - \phi_O(x_T, y_T)) \cdot (\phi_O(x_T, \bar{y}_T) - \phi_O(x_T, y_T)) \\
& + (\phi_E(x_T, \bar{y}_T) - \phi_E(x_T, y_T)) \cdot (\phi_E(x_T, \bar{y}_T) - \phi_E(x_T, y_T)) \\
& \leq w_O^{(T)} \cdot w_O^{(T)} + w_E^{(T)} \cdot w_E^{(T)} + 4C^2 \leq 4C^2 T \qquad \text{(A.5)} \\
& \therefore \left\|[w_O^{(T+1)}; w_E^{(T+1)}]\right\| \leq 2C\sqrt{T} \qquad \text{(A.6)}
\end{aligned}
$$

Eq. (A.5) follows from the fact that $s(x_T, y_T; w_O^{(T)}, w_E^{(T)}) > s(x_T, \bar{y}_T; w_O^{(T)}, w_E^{(T)})$ and $\left\|[\phi_O(x, y); \phi_E(x, y)]\right\|_2 \leq C$. Using equations (A.4) and (A.6) gives following bound on (A.3):

$$
\begin{aligned}
& \sum_{t=1}^{T} (s(x_t, \bar{y}_t; w_O^*, w_E^*) - s(x_t, y_t; w_O^*, w_E^*)) \\
& \qquad \leq 2C\sqrt{T} \left\|[w_O^*; w_E^*]\right\| \qquad \text{(A.7)}
\end{aligned}
$$

Assuming strictly α-informative feedback and re-writing equation (A.1) as:

$$
\begin{aligned}
& s(x_t, y_t^*; w_O^*, w_E^*) - s(x_t, y_t; w_O^*, w_E^*) \\
& \leq \frac{1}{\alpha}((s(x_t, \bar{y}_t; w_O^*, w_E^*) - s(x_t, y_t; w_O^*, w_E^*)) - \xi_t) \qquad \text{(A.8)}
\end{aligned}
$$

Combining equations (A.7) and (A.8) gives the bound on average regret (A.2).

A.2 Proof for Expected Regret

We now show the regret bounds for TPP under a weaker feedback assumption – expected α-informative feedback:

$$E_t[s(x_t, \bar{y}_t; w_O^*, w_E^*)] \geq s(x_t, y_t; w_O^*, w_E^*) + \alpha[s(x_t, y_t^*; w_O^*, w_E^*) - s(x_t, y_t; w_O^*, w_E^*)] - \xi_t$$

where the expectation is under choices $\bar{y}_t$ when y_t and x_t are known.

Corollary 2. *The expected regret of trajectory preference perceptron receiving expected α-informative feedback can be upper bounded for any $[w_O^*; w_E^*]$ as follows:*

$$E[REG_T] \leq \frac{2C \left\| [w_G^*; w_O^*] \right\|}{\alpha \sqrt{T}} + \frac{1}{\alpha T} \sum_{t=1}^{T} \bar{\xi}_t \tag{A.9}$$

Proof: Taking expectation on both sides of equation (A.3), (A.4) and (A.5) yields following equations respectively:

$$E[w_O^* \cdot w_O^{(T+1)} + w_E^* \cdot w_E^{(T+1)}] = \sum_{t=1}^{T} E[(s(x_t, \bar{y}_t; w_O^*, w_E^*) - s(x_t, y_t; w_O^*, w_E^*))] \tag{A.10}$$

$$E[w_O^* \cdot w_O^{(T+1)} + w_E^* \cdot w_E^{(T+1)}] \leq \left\| [w_O^*; w_E^*] \right\| E\left[\left\| [w_O^{(T+1)}; w_E^{(T+1)}] \right\| \right]$$

$$E[w_O^{(T+1)} \cdot w_O^{(T+1)} + w_E^{(T+1)} \cdot w_E^{(T+1)}] \leq 4C^2T$$

Applying Jensen's inequality on the concave function $\sqrt{\cdot}$ we get:

$$\begin{aligned}
&E[w_O^* \cdot w_O^{(T+1)} + w_E^* \cdot w_E^{(T+1)}] \\
&\leq \left\|[w_O^*; w_E^*]\right\| E\left[\left\|[w_O^{(T+1)}; w_E^{(T+1)}]\right\|\right] \\
&\leq \left\|[w_O^*; w_E^*]\right\| \sqrt{E[w_O^{(T+1)} \cdot w_O^{(T+1)} + w_E^{(T+1)} \cdot w_E^{(T+1)}]}
\end{aligned}$$

Using (A.10) gives the following bound:

$$\begin{aligned}
&\sum_{t=1}^{T} E[s(x_t, \bar{y}_t; w_O^*, w_E^*) - s(x_t, y_t; w_O^*, w_E^*)] \\
&\qquad\qquad \leq 2C\sqrt{T}\left\|[w_O^*; w_E^*]\right\|
\end{aligned}$$

Now using the fact that the user feedback is expected α-informative gives the regret bound (A.9).

APPENDIX B

CHAPTER 4 APPENDIX

B.1 Modeling Maneuvers with AIO-HMM

Given T seconds long driving context C before the maneuver $\mathcal{M}$, we learn a generative model for the context $P(C|\mathcal{M})$. The driving context C consists of the outside driving context and the inside driving context. The outside and inside contexts are temporal sequences represented by the outside features $\mathbf{x}_1^T = \{\mathbf{x}_1, .., \mathbf{x}_T\}$ and the inside features $\mathbf{z}_1^T = \{\mathbf{z}_1, .., \mathbf{z}_T\}$ respectively. The corresponding sequence of the driver's latent states is $h_1^T = \{h_1, .., h_T\}$. $\mathbf{x}$ and $\mathbf{z}$ are vectors and h is a discrete state.

$$\begin{aligned} P(C|\mathcal{M}) &= \sum_{h_1^T} P(\mathbf{z}_1^T, \mathbf{x}_1^T, h_1^T|\mathcal{M}) \\ &= P(\mathbf{x}_1^T|\mathcal{M}) \sum_{h_1^T} P(\mathbf{z}_1^T, h_1^T|\mathbf{x}_1^T, \mathcal{M}) \\ &\propto \sum_{h_1^T} P(\mathbf{z}_1^T, h_1^T|\mathbf{x}_1^T, \mathcal{M}) \end{aligned} \tag{B.1}$$

We model the correlations between $\mathbf{x}$, h and $\mathbf{z}$ with an AIO-HMM as shown in Figure 4.10. The AIO-HMM models the distribution in equation (B.1). It does not assume any generative process for the outside features $P(\mathbf{x}_1^T|\mathcal{M})$. It instead models them in a discriminative manner. The top (input) layer of the AIO-HMM consists of outside features $\mathbf{x}_1^T$. The outside features then affect the driver's latent states h_1^T, represented by the middle (hidden) layer, which then generates the inside features $\mathbf{z}_1^T$ at the bottom (output) layer. The events inside the vehicle such as the driver's head movements are temporally correlated because they are generally smooth. The AIO-HMM handles these dependencies with

autoregressive connections in the output layer.

Model Parameters. AIO-HMM has two types of parameters: (i) state transition parameters $\mathbf{w}$; and (ii) observation emission parameters ($\boldsymbol{\mu}$,$\boldsymbol{\Sigma}$). We use set $\mathcal{S}$ to denote the possible latent states of the driver. For each state $h = i \in \mathcal{S}$, we parametrize transition probabilities of leaving the state with log-linear functions, and parametrize the output layer feature emissions with normal distributions.

$$\text{Transition: } P(h_t = j | h_{t-1} = i, \mathbf{x}_t; \mathbf{w}_{ij}) = \frac{e^{\mathbf{w}_{ij} \cdot \mathbf{x}_t}}{\sum_{l \in \mathcal{S}} e^{\mathbf{w}_{il} \cdot \mathbf{x}_t}}$$

$$\text{Emission: } P(\mathbf{z}_t | h_t = i, \mathbf{x}_t, \mathbf{z}_{t-1}; \boldsymbol{\mu}_{it}, \boldsymbol{\Sigma}_i) = \mathcal{N}(\mathbf{z}_t | \boldsymbol{\mu}_{it}, \boldsymbol{\Sigma}_i)$$

The inside (vehicle) features represented by the output layer are jointly influenced by all three layers. These interactions are modeled by the mean and variance of the normal distribution. We model the mean of the distribution using the outside and inside features from the vehicle as follows:

$$\boldsymbol{\mu}_{it} = (1 + \mathbf{a}_i \cdot \mathbf{x}_t + \mathbf{b}_i \cdot \mathbf{z}_{t-1}) \boldsymbol{\mu}_i$$

In the equation above, $\mathbf{a}_i$ and $\mathbf{b}_i$ are parameters that we learn for every state $i \in \mathcal{S}$. Therefore, the parameters we learn for state $i \in \mathcal{S}$ are $\boldsymbol{\theta}_i = \{\boldsymbol{\mu}_i, \mathbf{a}_i, \mathbf{b}_i, \boldsymbol{\Sigma}_i$ and $\mathbf{w}_{ij} | j \in \mathcal{S}\}$, and the overall model parameters are $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_i | i \in \mathcal{S}\}$.

B.1.1 Learning AIO-HMM parameters

The training data $\mathcal{D} = \{(\mathbf{x}_{1,n}^{T_n}, \mathbf{z}_{1,n}^{T_n}) | n = 1, .., N\}$ consists of N instances of a maneuver $\mathcal{M}$. The goal is to maximize the data log-likelihood.

$$l(\boldsymbol{\Theta}; \mathcal{D}) = \sum_{n=1}^{N} \log P(\mathbf{z}_{1,n}^{T_n} | \mathbf{x}_{1,n}^{T_n}; \boldsymbol{\Theta}) \tag{B.2}$$

Directly optimizing equation (B.2) is challenging because parameters h representing the driver's states are *latent*. We therefore use the iterative EM procedure to learn the model parameters. In EM, instead of directly maximizing equation (B.2), we maximize its simpler lower bound. We estimate the lower bound in the E-step and then maximize that estimate in the M-step. These two steps are repeated iteratively.

E-step. In the E-step we get the lower bound of equation (B.2) by calculating the expected value of the *complete* data log-likelihood using the current estimate of the parameter $\hat{\boldsymbol{\Theta}}$.

$$\text{E-step: } Q(\boldsymbol{\Theta}; \hat{\boldsymbol{\Theta}}) = E[l_c(\boldsymbol{\Theta}; \mathcal{D}_c)|\hat{\boldsymbol{\Theta}}, \mathcal{D}] \tag{B.3}$$

where $l_c(\boldsymbol{\Theta}; \mathcal{D}_c)$ is the log-likelihood of the *complete* data $\mathcal{D}_c$ defined as:

$$\mathcal{D}_c = \{(\mathbf{x}_{1,n}^{T_n}, \mathbf{z}_{1,n}^{T_n}, h_{1,n}^{T_n})|n = 1, .., N\} \tag{B.4}$$

$$l_c(\boldsymbol{\Theta}; \mathcal{D}_c) = \sum_{n=1}^{N} \log P(\mathbf{z}_{1,n}^{T_n}, h_{1,n}^{T_n}|\mathbf{x}_{1,n}^{T_n}; \boldsymbol{\Theta}) \tag{B.5}$$

We should note that the occurrences of hidden variables h in $l_c(\boldsymbol{\Theta}; \mathcal{D}_c)$ are marginalized in equation (B.3), and hence h need not be known. We efficiently estimate $Q(\boldsymbol{\Theta}; \hat{\boldsymbol{\Theta}})$ using the forward-backward algorithm [164].

M-step. In the M-step we maximize the expected value of the complete data log-likelihood $Q(\boldsymbol{\Theta}; \hat{\boldsymbol{\Theta}})$ and update the model parameter as follows:

$$\text{M-step: } \boldsymbol{\Theta} = \arg\max_{\boldsymbol{\Theta}} Q(\boldsymbol{\Theta}; \hat{\boldsymbol{\Theta}}) \tag{B.6}$$

Solving equation (B.6) requires us to optimize for the parameters $\boldsymbol{\mu}$, $\mathbf{a}$, $\mathbf{b}$, $\boldsymbol{\Sigma}$ and $\mathbf{w}$. We optimize all parameters expect $\mathbf{w}$ exactly by deriving their closed form update expressions. We optimize $\mathbf{w}$ using the gradient descent.

B.1.2 Inference of Maneuvers

Our learning algorithm trains separate AIO-HMM models for each maneuver. The goal during inference is to determine which model best explains the past T seconds of the driving context not seen during training. We evaluate the likelihood of the inside and outside feature sequences ($\mathbf{z}_1^T$ and $\mathbf{x}_1^T$) for each maneuver, and anticipate the probability $P_{\mathcal{M}}$ of each maneuver $\mathcal{M}$ as follows:

$$P_{\mathcal{M}} = P(\mathcal{M}|\mathbf{z}_1^T, \mathbf{x}_1^T) \propto P(\mathbf{z}_1^T, \mathbf{x}_1^T|\mathcal{M})P(\mathcal{M}) \tag{B.7}$$

Algorithm 5 shows the complete inference procedure. The inference in equation (B.7) simply requires a forward-pass [164] of the AIO-HMM, the complexity of which is $O(T(|\mathcal{S}|^2 + |\mathcal{S}||\mathbf{z}|^3 + |\mathcal{S}||\mathbf{x}|))$. However, in practice it is only $O(T|\mathcal{S}||\mathbf{z}|^3)$ because $|\mathbf{z}|^3 \gg |S|$ and $|\mathbf{z}|^3 \gg |\mathbf{x}|$. Here $|\mathcal{S}|$ is the number of discrete states representing the driver's intention, while $|\mathbf{z}|$ and $|\mathbf{x}|$ are the dimensions of the inside and outside feature vectors respectively. In equation (B.7) $P(\mathcal{M})$ is the prior probability of maneuver $\mathcal{M}$. We assume an uninformative uniform prior over the maneuvers.

Algorithm 5: Anticipating maneuvers

input Driving videos, GPS, Maps and Vehicle Dynamics

output Probability of each maneuver

Initialize the face tracker with the driver's face

while *driving* **do**

Track the driver's face [238]

Extract features $\mathbf{z}_1^T$ and $\mathbf{x}_1^T$ (Sec. 4.7)

Inference $P_{\mathcal{M}} = P(\mathcal{M}|\mathbf{z}_1^T, \mathbf{x}_1^T)$ (Eq. (B.7))

Send the inferred probability of each maneuver to ADAS

end while

BIBLIOGRAPHY

[1] Bosch urban. `http://bit.ly/1feM3JM`. Accessed: 2015-04-23.

[2] B. Akgun, M. Cakmak, K. Jiang, and A. L. Thomaz. Keyframe-based learning from demonstration. *International Journal of Social Robotics*, 4(4):343–355, 2012.

[3] R. Alterovitz, T. Siméon, and K. Goldberg. The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. In *Proceedings of Robotics: Science and Systems*, 2007.

[4] A. Anand, H. S. Koppula, T. Joachims, and A. Saxena. Contextually guided semantic labeling and search for 3d point clouds. *International Journal of Robotics Research*, 2012.

[5] R. D. Andrea. Guest editorial: A revolution in the warehouse: A retrospective on kiva systems and the grand challenges ahead. *IEEE Tran. on Automation Science and Engineering (T-ASE)*, 9(4), 2012.

[6] A. Andreas, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[7] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[8] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. *Dbpedia: A nucleus for a web of open data*. Springer, 2007.

[9] T. Baltrusaitis, P. Robinson, and L-P. Morency. Constrained local neural fields for robust facial landmark detection in the wild. In *ICCV Workshop*, 2013.

[10] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[11] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mosenlechner, D. Pangercic, T. Ruhr, and M. Tenorth. Robotic roommates making pancakes. In

Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on, pages 529–536. IEEE, 2011.

[12] M. Beetz, M. Tenorth, and J. Winkler. open-ease a knowledge processing service for robots and robotics/ai researchers. *TZI Technical Report*, 74, 2014.

[13] Y. Bengio and O. Delalleau. On the expressive power of deep architectures. In *Algorithmic Learning Theory*, pages 18–36, 2011.

[14] Y. Bengio and O. Frasconi. An input output hmm architecture. *Advances in Neural Information Processing Systems*, 1995.

[15] Y. Bengio, Y. LeCun, and D. Henderson. Globally trained handwritten word recognizer using spatial representation, convolutional neural networks, and hidden markov models. *Advances in Neural Information Processing Systems*, 1994.

[16] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the International Conference on Machine Learning*, 2009.

[17] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun. Learning motion patterns of people for compliant robot motion. *International Journal of Robotics Research*, 2005.

[18] D. Berenson, P. Abbeel, and K. Goldberg. A robot path planning framework that learns from experience. In *Proceedings of the International Conference on Robotics and Automation*, 2012.

[19] J. V. D. Berg, P. Abbeel, and K. Goldberg. Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information. In *Proceedings of Robotics: Science and Systems*, June 2010.

[20] H. Berndt, J. Emmert, and K. Dietmayer. Continuous driver intention recognition with hidden markov models. In *IEEE Intelligent Transportation Systems Conference*, 2008.

[21] S. Bhattacharya, M. Likhachev, and V. Kumar. Identification and representation of homotopy classes of trajectories for search-based path planning in 3d. In *Proceedings of Robotics: Science and Systems*, 2011.

[22] R. Bischoff, A. Kazi, and M. Seyfarth. The morpha style guide for icon-based programming. In *Proceedings. 11th IEEE International Workshop on RHIC.*, 2002.

[23] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM SIGMOD Management Of Data*, pages 1247–1250, 2008.

[24] M. Bollini, S. Tellex, T. Thompson, M. Roy, and D. Rus. Interpreting and executing recipes with a cooking robot. In *ISER*, 2012.

[25] L. Bottou, Y. Bengio, and Y. LeCun. Global training of document processing systems using graph transformer networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1997.

[26] W. Brendel and S. Todorovic. Learning spatiotemporal graphs of human activities. In *Proceedings of the International Conference on Computer Vision*, 2011.

[27] W. Byeon, T.M. Breuel, F. Raue, and M. Liwicki. Scene labeling with lstm recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[28] M. Cakmak, S. S. Srinivasa, M. K. Lee, J. Forlizzi, and S.B. Kiesler. Human preferences for robot-human hand-over configurations. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, 2011.

[29] S. Calinon, F. Guenter, and A. Billard. On learning, representing, and generalizing a task in a humanoid robot. *Sys., Man, and Cybernetics, Part B: Cybernetics, IEEE Trans. on*, 2007.

[30] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, volume 5, page 3, 2010.

[31] L. C. Chen, G. Papandreou, I. Kokkinos, and K. Murphyand A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv:1412.7062*, 2014.

[32] L. C. Chen, A. Schwing, A. L. Yuille, and R. Urtasun. Learning deep

structured models. In *Proceedings of the International Conference on Machine Learning*, 2015.

[33] M. Chen and A. Hauptmann. Mosift: Recognizing human actions in surveillance videos. *CMU Tech Report*, 2009.

[34] X. Chen, A. Shrivastava, and A. Gupta. NEIL: Extracting Visual Knowledge from Web Data. In *ICCV*, 2013.

[35] X. Chen and C. L. Zitnick. Mind's eye: A recurrent visual representation for image caption generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[36] M. J-Y. Chung, M. Forbes, M. Cakmak, and R. Rao. Accelerating imitation learning through crowdsourcing. In *Proceedings of the International Conference on Robotics and Automation*, 2014.

[37] B. J. Cohen, S. Chitta, and M. Likhachev. Search-based planning for manipulation with motion primitives. In *Proceedings of the International Conference on Robotics and Automation*, pages 2902–2908, 2010.

[38] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6), 2001.

[39] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3), 1995.

[40] A. Criminisi, J. Shotton, and E. Konukoglu. Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning. *MSR TR*, 5(6), 2011.

[41] N. Curtis and J. Xiao. Efficient and effective grasping of novel objects through learning and adapting a knowledge base. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, 2008.

[42] Dave D. Ferguson and A. Stentz. Anytime rrts. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, 2006.

[43] Hal Daumé Iii, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

[44] Y. N. Dauphin, H. de Vries, J. Chung, and Y. Bengio. Rmsprop and equilibrated adaptive learning rates for non-convex optimization. *arXiv:1502.04390*, 2015.

[45] V. Delaitre, D. Fouhey, I. Laptev, J. Sivic, A. Gupta, and A. Efros. Scene semantics from long-term observation of people. In *Proceedings of the European Conference on Computer Vision*, 2012.

[46] J. Deng, W. Dong, R. Socher, L-J Li, K. Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[47] D. Dey, T. Y. Liu, M. Hebert, and J. A. Bagnell. Contextual sequence prediction with application to control library optimization. In *Proceedings of Robotics: Science and Systems*, 2012.

[48] R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, CMU, RI, August 2010.

[49] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[50] X.L. Dong, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the ACM Special Interest Group on Knowledge Discovery and Data Mining*, 2014.

[51] A. Doshi, B. Morris, and M. M. Trivedi. On-road prediction of driver's intent with multimodal sensory cues. *IEEE Pervasive Computing*, 2011.

[52] B. Douillard, D. Fox, and F. Ramos. A spatio-temporal probabilistic model for multi-sensor multi-class object recognition. In *Robotics Research*, 2011.

[53] A. Dragan, K. Lee, and S. Srinivasa. Legibility and predictability of robot motion. In *Human Robot Interaction*, 2013.

[54] A. Dragan and S. Srinivasa. Formalizing assistive teleoperation. In *Proceedings of Robotics: Science and Systems*, 2012.

[55] A. Dragan and S. Srinivasa. Generating legible motion. In *Proceedings of Robotics: Science and Systems*, 2013.

[56] K. Driggs-Campbell, V. Shia, and R. Bajcsy. Improved driver modeling for human-in-the-loop vehicular control. In *Proceedings of the International Conference on Robotics and Automation*, 2015.

[57] Y. Du, W. Wang, and L. Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[58] E. Şahin, M. Çakmak, M. R. Doğar, E. Uğur, and G. Üçoluk. To afford or not to afford: A new formalization of affordances toward affordance-based robot control. *Adaptive Behavior*, 15(4):447–472, 2007.

[59] L. H. Erickson and S. M. LaValle. Survivability: Measuring and ensuring path diversity. In *Proceedings of the International Conference on Robotics and Automation*, 2009.

[60] A. Ettlin and H. Bleuler. Randomised rough-terrain robot motion planning. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, 2006.

[61] C. Fellbaum. *WordNet*. Wiley Online Library, 1998.

[62] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[63] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.

[64] D. A. Ferrucci. Introduction to this is watson. *IBM J. of RnD*, 56(3.4):1–1, 2012.

[65] L. Fletcher, N. Apostoloff, L. Petersson, and A. Zelinsky. Vision in and out of vehicles. *IEEE IS*, 18(3), 2003.

[66] L. Fletcher, G. Loy, N. Barnes, and A. Zelinsky. Correlating driver gaze with the road scene for driver assistance systems. *Robotics and Autonomous Systems*, 52(1), 2005.

[67] K. Fragkiadaki, S. Levine, P. Felsen, and J. Malik. Recurrent network models for human dynamics. In *Proceedings of the International Conference on Computer Vision*, 2015.

[68] B. Frohlich, M. Enzweiler, and U. Franke. Will this car change the lane?turn signal recognition in the frequency domain. In *IEEE International Vehicle Symposium Proceedings*, 2014.

[69] A. G. Schwing G and R. Urtasun. Fully connected deep structured networks. *arXiv:1503.02351*, 2015.

[70] J. J. Gibson. *The ecological approach to visual perception*. Routledge, 1986.

[71] M. J. Gielniak, C. Karen Liu, and A. L. Thomaz. Generating human-like motion for robots. *International Journal of Robotics Research*, 32(11), 2013.

[72] R. Girshick. Fast r-cnn. In *Proceedings of the International Conference on Computer Vision*, 2015.

[73] C. Goller and A. Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Neural Networks, IEEE*, volume 1, 1996.

[74] D. Gossow, A. Leeperand D. Hershberger, and M. Ciocarlie. Interactive markers: 3-d user interfaces for ros applications [ros topics]. *Robotics & Automation Magazine, IEEE*, 18(4):14–15, 2011.

[75] A. Graves. Generating sequences with recurrent neural networks. *arXiv:1308.0850*, 2013.

[76] A. Graves and N. Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*, 2014.

[77] C. J. Green and A. Kelly. Toward optimal sampling in the space of paths. In *Robotics Research*, 2011.

[78] S. Guadarrama, E. Rodner, K. Saenko, N. Zhang, R. Farrell, J. Donahue, and T. Darrell. Open-vocabulary object retrieval. In *Proceedings of Robotics: Science and Systems*, 2014.

[79] A. Gupta, A. Kembhavi, and L. S. Davis. Observing human-object interactions: Using spatial and functional compatibility for recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(10), 2009.

[80] S. Gupta, R. Girshick, P. Arbelaez, and J. Malik. Learning rich features from RGB-D images for object detection and segmentation. In *Proceedings of the European Conference on Computer Vision*, 2014.

[81] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, and A. Coates. Deepspeech: Scaling up end-to-end speech recognition. *arXiv:1412.5567*, 2014.

[82] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1), 2004.

[83] S. El Hihi and Y. Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *NIPS*, 1995.

[84] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8), 1997.

[85] J. Hoffart, F. M Suchanek, K. Berberich, and G. Weikum. Yago2: a spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28–61, 2013.

[86] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7), 2014.

[87] M. E. Jabon, J. N. Bailenson, E. Pontikakis, L. Takayama, and C. Nass. Facial expression analysis for predicting unsafe driving behavior. *IEEE Pervasive Computing*, 2010.

[88] L. Jaillet, J. Cortés, and T. Siméon. Sampling-based path planning on configuration-space costmaps. *IEEE Transactions on Robotics*, 26(4), 2010.

[89] A. Jain. Neuralmodels. `https://github.com/asheshjain399/NeuralModels`, 2015.

[90] A. Jain, D. Das, J. Gupta, and A. Saxena. Planit: A crowdsourcing approach for learning to plan paths from large scale preference feedback. In *Proceedings of the International Conference on Robotics and Automation*, 2015.

[91] A. Jain, H. S. Koppula, B. Raghavan, S. Soh, and A. Saxena. Car that knows before you do: Anticipating maneuvers via learning temporal driving models. In *Proceedings of the International Conference on Computer Vision*, 2015.

[92] A. Jain, H. S. Koppula, S. Soh, B. Raghavan, A. Singh, and A. Saxena. Brain4cars: Car that knows before you do via sensory-fusion deep learning architecture. *CoRR*, abs/1601.00740, 2016.

[93] A. Jain, S. Sharma, T. Joachims, and A. Saxena. Learning preferences for manipulation tasks from online coactive feedback. *International Journal of Robotics Research*, 2015.

[94] A. Jain, S. Sharma, and A. Saxena. Beyond geometric path planning: Learning context-driven user preferences via sub-optimal feedback. In *Proceedings of the International Symposium on Robotics Research*, 2013.

[95] A. Jain, A. Singh, H. S. Koppula, S. Soh, and A. Saxena. Recurrent neural networks for driver activity anticipation via sensory-fusion architecture. In *Proceedings of the International Conference on Robotics and Automation*, 2016.

[96] A. Jain, B. Wojcik, T. Joachims, and A. Saxena. Learning trajectory preferences for manipulators via iterative improvement. In *Advances in Neural Information Processing Systems*, 2013.

[97] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[98] M. Jain, J. C. van Gemert, T. Mensink, and C. Snoek. Objects2action: Classifying and localizing actions without any video example. In *Proceedings of the International Conference on Computer Vision*, 2015.

[99] Y. Jiang, H. Koppula, and A. Saxena. Hallucinated humans as the hidden context for labeling 3d scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[100] Y. Jiang, M. Lim, and A. Saxena. Learning object arrangements in 3d scenes using human context. In *Proceedings of the International Conference on Machine Learning*, 2012.

[101] Y. Jiang, M. Lim, C. Zheng, and A. Saxena. Learning to place new objects in a scene. *International Journal of Robotics Research*, 31(9):1021–1043, 2012.

[102] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the ACM Special Interest Group on Knowledge Discovery and Data Mining*, 2002.

[103] T. Joachims. Training linear svms in linear time. In *Proceedings of the ACM Special Interest Group on Knowledge Discovery and Data Mining*, 2006.

[104] T. Joachims, T. Finley, and C.-N. J. Yu. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, 2009.

[105] T. Joachims, L. Granka, B. Pan, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the ACM SIGIR Conference on Information Retrieval*, 2005.

[106] Z. Kalal, K. Mikolajczyk, and J. Matas. Forward-backward error: Automatic detection of tracking failures. In *Proceedings of the International Conference on Pattern Recognition*, 2010.

[107] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Proceedings of Robotics: Science and Systems*, 2010.

[108] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, 2011.

[109] A. Karpathy, J. Johnson, and F. F. Li. Visualizing and understanding recurrent networks. *arXiv:1506.02078*, 2015.

[110] D. Katz, Y. Pyuro, and O. Brock. Learning to manipulate articulated objects in unstructured environments using a grounded relational representation. In *Proceedings of Robotics: Science and Systems*, 2008.

[111] D. Katz, A. Venkatraman, M. Kazemi, J. A. Bagnell, and A. Stentz. Perceiving, learning, and exploiting object affordances for autonomous pile manipulation. In *Proceedings of Robotics: Science and Systems*, 2013.

[112] D. Katz, A. Venkatraman, M. Kazemi, J. A. Bagnell, and A. Stentz. Perceiving, learning, and exploiting object affordances for autonomous pile manipulation. *Autonomous Robots*, 37(4), 2014.

[113] K. M. Kitani, B. D. Ziebart, J. A. Bagnell, and M. Hebert. Activity forecasting. In *Proceedings of the European Conference on Computer Vision*, 2012.

[114] E. Klingbeil, D. Rao, B. Carpenter, V. Ganapathi, A. Y. Ng, and O. Khatib. Grasping with application to an autonomous checkout robot. In *Proceedings of the International Conference on Robotics and Automation*, 2011.

[115] R. A. Knepper, T. Layton, J. Romanishin, and D. Rus. Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In *Proceedings of the International Conference on Robotics and Automation*, 2013.

[116] J. Kober and J. Peters. Policy search for motor primitives in robotics. *ML*, 84(1), 2011.

[117] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, 2004.

[118] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[119] H. Koppula, R. Gupta, and A. Saxena. Learning human activities and object affordances from rgb-d videos. *International Journal of Robotics Research*, 32(8), 2013.

[120] H. Koppula, A. Jain, and A. Saxena. Anticipatory planning for human-robot teams. In *ISER*, 2014.

[121] H. Koppula and A. Saxena. Anticipating human activities using object affordances for reactive robotic response. In *Proceedings of Robotics: Science and Systems*, 2013.

[122] H. Koppula and A. Saxena. Learning spatio-temporal structure from rgb-d videos for human activity detection and anticipation. In *Proceedings of the International Conference on Machine Learning*, 2013.

[123] H. Koppula and A. Saxena. Anticipating human activities using object af-

fordances for reactive robotic response. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015.

[124] H.S. Koppula, A. Anand, T. Joachims, and A. Saxena. Semantic labeling of 3d point clouds for indoor scenes. In *Advances in Neural Information Processing Systems*, 2011.

[125] H.S. Koppula and A. Saxena. Physically grounded spatio-temporal object affordances. In *Proceedings of the European Conference on Computer Vision*, 2013.

[126] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 2009.

[127] P. Krähenbühl and V. Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. *arXiv:1210.5644*, 2012.

[128] H. Kretzschmar, M. Kuderer, and W. Burgard. Learning to predict trajectories of cooperatively navigating agents. In *Proceedings of the International Conference on Robotics and Automation*, 2014.

[129] F. R. Kschischang, B. J. Frey, and H-A. Loeliger. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Trans.*, 47(2), 2001.

[130] M. Kuderer, H. Kretzschmar, C. Sprunk, and W. Burgard. Feature-based prediction of trajectories for socially compliant navigation. In *Proceedings of Robotics: Science and Systems*, 2012.

[131] N. Kuge, T. Yamamura, O. Shimoyama, and A. Liu. A driver behavior recognition method based on a driver model framework. Technical report, SAE Technical Paper, 2000.

[132] P. Kumar, M. Perrollaz, S. Lefevre, and C. Laugier. Learning-based approach for online lane change intention prediction. In *IEEE International Vehicle Symposium Proceedings*, 2013.

[133] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*, 2001.

[134] K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view

RGB-D object dataset. In *Proceedings of the International Conference on Robotics and Automation*, 2011.

[135] I. Laptev, M. Marszałek, C. Schmid, and B. Rozenfeld. Learning realistic human actions from movies. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[136] C. Laugier, I. E. Paromtchik, M. Perrollaz, MY. Yong, J-D. Yoder, C. Tay, K. Mekhnacha, and A. Negre. Probabilistic analysis of dynamic scenes and collision risks assessment to improve driving safety. *ITS Magazine, IEEE*, 3(4), 2011.

[137] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5), May 2001.

[138] D. B Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Commun. ACM*, 38(11):33–38, 1995.

[139] I. Lenz, H. Lee, and A. Saxena. Deep learning for detecting robotic grasps. In *Proceedings of Robotics: Science and Systems*, 2013.

[140] P. Leven and S. Hutchinson. Using manipulability to bias sampling during the construction of probabilistic roadmaps. *IEEE Trans. on Robotics and Automation*, 19(6), 2003.

[141] S. Levine and V. Koltun. Continuous inverse optimal control with locally optimal examples. In *Proceedings of the International Conference on Machine Learning*, 2012.

[142] J. Lezama, K. Alahari, J. Sivic, and I. Laptev. Track to the future: Spatio-temporal video segmentation with long-range motion cues. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[143] S. Li, W. Zhang, and A. B. Chan. Maximum-margin structured learning with deep networks for 3d human pose estimation. In *Proceedings of the International Conference on Computer Vision*, 2015.

[144] Y. Li and R. Nevatia. Key object driven multi-category object recognition, localization and tracking using spatio-temporal context. In *Proceedings of the European Conference on Computer Vision*, 2008.

[145] M. Liebner, M. Baumann, F. Klanner, and C. Stiller. Driver intent inference at urban intersections using the intelligent driver model. In *IEEE International Vehicle Symposium Proceedings*, 2012.

[146] G. Lin, C. Shen, I. Reid, et al. Efficient piecewise training of deep structured models for semantic segmentation. *arXiv:1504.01013*, 2015.

[147] Z. Liu, X. Li, P. Luo, C. C. Loy, and X. Tang. Semantic image segmentation via deep parsing network. In *Proceedings of the International Conference on Computer Vision*, 2015.

[148] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1981.

[149] J. Mainprice and D. Berenson. Human-robot collaborative manipulation planning using early prediction of human motion. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, 2013.

[150] J. Mainprice, E. A. Sisbot, L. Jaillet, J. Cortés, R. Alami, and T. Siméon. Planning human-aware motions using a sampling-based costmap planner. In *Proceedings of the International Conference on Robotics and Automation*, 2011.

[151] J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *Proceedings of the International Conference on Robotics and Automation*, 2010.

[152] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, 2008.

[153] I. Matthews and S. Baker. Active appearance models revisited. *International Journal of Computer Vision*, 60(2), 2004.

[154] A. McCallum, K. Schultz, and S. Singh. Factorie: Probabilistic programming via imperatively defined factor graphs. In *Advances in Neural Information Processing Systems*, 2009.

[155] C. McManus, B. Upcroft, and P. Newman. Scene signatures: Localised and point-less features for localisation. In *Proceedings of Robotics: Science and Systems*, 2014.

[156] T. Mikolov, A. Joulin, S. Chopra, M. Mathieu, and M. A. Ranzato. Learning longer memory in recurrent neural networks. *arXiv:1412.7753*, 2014.

[157] A. T. Miller and P. K. Allen. Graspit! a versatile simulator for robotic grasping. *Robotics & Automation Magazine, IEEE*, 11(4), 2004.

[158] D. K. Misra, J. Sung, K. Lee, and A. Saxena. Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. *Proceedings of Robotics: Science and Systems*, 2014.

[159] D.K. Misra, J. Sung, K. Lee, and A. Saxena. Tell me dave: Context-sensitive grounding of natural language to mobile manipulation instructions. In *Proceedings of Robotics: Science and Systems*, 2014.

[160] A. R. Mohamed, T. N. Sainath, G. Dahl, B. Ramabhadran, G. E. Hinton, and M. A. Picheny. Deep belief networks using discriminative features for phone recognition. In *(ICASSP)*, pages 5060–5063, 2011.

[161] L. Montesano, M. Lopes, A. Bernardino, and J. S.-Victor. Learning object affordances: from sensory–motor coordination to imitation. *IEEE Transactions on Robotics*, 2008.

[162] L. Morency, A. Quattoni, and T. Darrell. Latent-dynamic discriminative models for continuous gesture recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[163] B. Morris, A. Doshi, and M. Trivedi. Lane change intent prediction for driver assistance: On-road design and evaluation. In *IEEE International Vehicle Symposium Proceedings*, 2011.

[164] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[165] T. Naseer, L. Spinello, W. Burgard, and C. Stachniss. Robust visual robot localization across seasons using network flows. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, 2014.

[166] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. Multimodal deep learning. In *Proceedings of the International Conference on Machine Learning*, 2011.

[167] NHTSA. 2012 motor vehicle crashes: overview. *N. Highway Traffic Safety Administration, Washington, D.C., Tech. Rep.*, 2013.

[168] S. Nikolaidis, P. Lasota, G. Rossano, C. Martinez, T. Fuhlbrigge, and J. Shah. Human-robot collaboration in manufacturing: Quantitative evaluation of predictable, convergent joint action. In *International Symposium on Robotics*, 2013.

[169] S. Nikolaidis and J. Shah. Human-robot teaming using shared mental models. In *HRI, Workshop on Human-Agent-Robot Teamwork*, 2012.

[170] S. Nikolaidis and J. Shah. Human-robot cross-training: Computational formulation, modeling and evaluation of a human team training strategy. In *IEEE/ACM ICHRI*, 2013.

[171] N. Oliver and A. P. Pentland. Graphical models for driver behavior recognition in a smartcar. In *IEEE International Vehicle Symposium Proceedings*, 2000.

[172] p. abbeel, a. coates, and a. y. ng. autonomous helicopter aerobatics through apprenticeship learning. *International Journal of Robotics Research*, 29(13), 2010.

[173] R. Paolini, A. Rodriguez, S. S. Srinivasa, and M. T. Mason. A data-driven statistical framework for post-grasp manipulation. In *ISER*, 2013.

[174] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *arXiv:1211.5063*, 2012.

[175] M. Phillips, B. Cohen, S. Chitta, and M. Likhachev. E-graphs: Bootstrapping planning with experience graphs. In *Proceedings of Robotics: Science and Systems*, 2012.

[176] A. Pieropan, C. H. Ek, and H. Kjellström. Recognizing object affordances in terms of spatio-temporal object-object relationships. In *IEEE-RAS Intl. Conf. on Humanoid Robots*, 2014.

[177] A. Quattoni, M. Collins, and T. Darrell. Conditional random fields for object recognition. In *Advances in Neural Information Processing Systems*, 2004.

[178] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[179] K. Raman and T. Joachims. Learning socially optimal information systems from egoistic users. In *Proceedings of the European Conference on Machine Learning*, 2013.

[180] N. Ratliff. *Learning to Search: Structured Prediction Techniques for Imitation Learning*. PhD thesis, CMU, RI, 2009.

[181] N. Ratliff, J. A. Bagnell, and M. Zinkevich. Maximum margin planning. In *Proceedings of the International Conference on Machine Learning*, 2006.

[182] N. Ratliff, J. A. Bagnell, and M. Zinkevich. (online) subgradient methods for structured prediction. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2007.

[183] N. Ratliff, D. Silver, and J. A. Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53, 2009.

[184] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *Proceedings of the International Conference on Robotics and Automation*, 2009.

[185] M. Rezaei and R. Klette. Look at the driver, look at the road: No distraction! no accident! In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[186] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1-2), 2006.

[187] J. Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.

[188] S. Ross, J. Zhou, Y. Yue, D. Dey, and J. A. Bagnell. Learning policies for contextual submodular prediction. *Proceedings of the International Conference on Machine Learning*, 2013.

[189] M. Rouhizadeh, D. Bauer, R. E. Coyne, O. C. Rambow, and R. Sproat. Collecting spatial information for locations in a text-to-scene conversion system. *Computational Models for Spatial Languages*, 2011.

[190] T. Rueda-Domingo, P. Lardelli-Claret, J. Luna del Castillo, J. Jimenez-Moleon, M. Garcia-Martin, and A. Bueno-Cavanillas. The influence of

passengers on the risk of the driver causing a car collision in spain: Analysis of collisions from 1990 to 1999. *Accident Analysis & Prevention*, 2004.

[191] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: a database and web-based tool for image annotation. *IJCV*, 77(1-3), 2008.

[192] M. S. Ryoo. Human activity prediction: Early recognition of ongoing activities from streaming videos. In *Proceedings of the International Conference on Computer Vision*, 2011.

[193] A. Saxena, J. Driemeyer, and A. Y. Ng. Robotic grasping of novel objects using vision. *International Journal of Robotics Research*, 27(2):157–173, 2008.

[194] A. Saxena, A. Jain, O. Sener, A. Jami, D. K. Misra, and H. S. Koppula. Robo brain: Large-scale knowledge engine for robots. In *Proceedings of the International Symposium on Robotics Research*, 2015.

[195] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Proceedings of Robotics: Science and Systems*, 2013.

[196] J. Shi and C. Tomasi. Good features to track. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1994.

[197] Q. Shi, L. Cheng, L. Wang, and A. Smola. Human action segmentation and recognition using discriminative semi-markov models. *International Journal of Computer Vision*, 93(1), 2011.

[198] V. Shia, Y. Gao, R. Vasudevan, K. D. Campbell, T. Lin, F. Borrelli, and R. Bajcsy. Semiautonomous vehicular control using driver modeling. *IEEE Transactions on Intelligent Transportation Systems*, 15(6), 2014.

[199] P. Shivaswamy and T. Joachims. Online structured prediction via coactive learning. In *Proceedings of the International Conference on Machine Learning*, 2012.

[200] B. Shneiderman and C. Plaisant. *Designing The User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley Publication, 2010.

[201] D. Silver, J. A. Bagnell, and A. Stentz. Learning from demonstration for autonomous navigation in complex unstructured terrain. *International Journal of Robotics Research*, 2010.

[202] E. A. Sisbot and R. Alami. A human-aware manipulation planner. *Robotics, IEEE Transactions on*, 28, 2012.

[203] E. A. Sisbot, L. F. Marin, and R. Alami. Spatial reasoning for human robot interaction. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, 2007.

[204] E. A. Sisbot, L. F. Marin-Urias, R. Alami, and T. Simeon. A human aware mobile robot motion planner. *IEEE Transactions on Robotics*, 2007.

[205] R. Socher, C. C. Lin, A. Y. Ng, and C. D. Manning. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In *Proceedings of the International Conference on Machine Learning*, 2011.

[206] S. Sra. A short note on parameter approximation for von mises-fisher distributions: and a fast implementation of i s (x). *Computational Statistics*, 27(1), 2012.

[207] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using lstms. In *Proceedings of the International Conference on Machine Learning*, 2015.

[208] A. Stopp, S. Horstmann, S. Kristensen, and F. Lohnert. Towards interactive learning for manufacturing assistants. In *Proceedings. 10th IEEE International Workshop on RHIC.*, 2001.

[209] I. A. Sucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012. `http://ompl.kavrakilab.org`.

[210] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *Proceedings of the International Conference on World Wide Web*, 2007.

[211] C. Sun and R. Nevatia. Active: Activity concept transitions in video event classification. In *Proceedings of the International Conference on Computer Vision*, 2013.

[212] J. Sung, S. H. Jin, and A. Saxena. Robobarista: Object part-based transfer of manipulation trajectories from crowd-sourcing in 3d pointclouds. In *Proceedings of the International Symposium on Robotics Research*, 2015.

[213] I. Sutskever, G. Hinton, and G. Taylor. The recurrent temporal restricted boltzmann machine. In *Advances in Neural Information Processing Systems*, 2009.

[214] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, 2014.

[215] C. Sutton and A. McCallum. An introduction to conditional random fields. *Machine Learning*, 4(4), 2011.

[216] K. Tamane, M. Revfi, and T. Asfour. Synthesizing object receiving motions of humanoid robots with human motion database. In *Proceedings of the International Conference on Robotics and Automation*, 2013.

[217] B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2002.

[218] A. Tawari, S. Sivaraman, M. Trivedi, T. Shannon, and M. Tippelhofer. Looking-in and looking-out vision for urban intelligent assistance: Estimation of driver attentive state and dynamic surround for safe merging and braking. In *IEEE IVS*, 2014.

[219] G. Taylor and G. E. Hinton. Factored conditional restricted boltzmann machines for modeling motion style. In *Proceedings of the International Conference on Machine Learning*, 2009.

[220] G. Taylor, L. Sigal, D. J. Fleet, and G. E. Hinton. Dynamical binary latent variable models for 3d human pose tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[221] G. W. Taylor, G. E. Hinton, and S. T. Roweis. Modeling human motion using binary latent variables. In *Advances in Neural Information Processing Systems*, 2006.

[222] S. Tellex, R. Knepper, A. Li, D. Rus, and N. Roy. Asking for help using inverse semantics. In *Proceedings of Robotics: Science and Systems*, 2014.

[223] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. J. Teller, and N. Roy. Understanding natural language commands for robotic nav-

igation and mobile manipulation. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, 2011.

[224] M. Tenorth, D. Nyga, and M. Beetz. Understanding and executing instructions for everyday manipulation tasks from the world wide web. In *ICRA*, 2010.

[225] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT press, 2005.

[226] M. E. Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1, 2001.

[227] C. Tomasi and T. Kanade. Detection and tracking of point features. *International Journal of Computer Vision*, 1991.

[228] J. Tompson, M. Stein, Y. Lecun, and K. Perlin. Real-time continuous pose recovery of human hands using convolutional networks. *ACM TOG*, 33(5), 2014.

[229] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the International Conference on Computer Vision*, 2015.

[230] M. Trivedi, T. Gandhi, and J. McCall. Looking-in and looking-out of a vehicle: Computer-vision-based enhanced vehicle safety. *IEEE Transactions on Intelligent Transportation Systems*, 8(1), 2007.

[231] E. Ugur, E. Oztop, and E. Sahin. Goal emulation and planning in perceptual space using learned affordances. *Robotics and Autonomous Systems*, 2011.

[232] R. Urtasun, D. J. Fleet, A. Geiger, J. Popović, T. J. Darrell, and N. D. Lawrence. Topologically-constrained latent variable models. In *Proceedings of the International Conference on Machine Learning*, 2008.

[233] K. F. Uyanik, Y. Caliskan, A. K. Bozcuoglu, S. Kalkan O. Yuruten, and E. Sahin. Learning social affordances and using them for planning. In *CogSys*, 2013.

[234] D. L. Vail, M. M. Veloso, and J. D. Lafferty. Conditional random fields for activity recognition. In *AAMAS*, 2007.

[235] R. Vasudevan, V. Shia, Y. Gao, R. Cervera-Navarro, R. Bajcsy, and F. Borrelli. Safe semi-autonomous control with enhanced driver modeling. In *American Control Conference*, 2012.

[236] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, and K. Saenko. Translating videos to natural language using deep recurrent neural networks. *arXiv:1412.4729*, 2014.

[237] P. Vernaza and J. A. Bagnell. Efficient high dimensional maximum entropy modeling via symmetric partition functions. In *Advances in Neural Information Processing Systems*, 2012.

[238] P. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2), 2004.

[239] M. Waibel, M. Beetz, R. D'Andrea, et al. Roboearth: A world wide web for robots. *IEEE R & A. Magz.*, 2011.

[240] H. Wang and C. Schmid. Action recognition with improved trajectories. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[241] J. M. Wang, D. J. Fleet, and A. Hertzmann. Gaussian process dynamical models for human motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2), 2008.

[242] S. B. Wang, A. Quattoni, L. Morency, D. Demirdjian, and T. Darrell. Hidden conditional random fields for gesture recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[243] Y. Wang and Q. Ji. A dynamic conditional random field model for object segmentation in image sequences. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[244] Z. Wang, K. Mülling, M. Deisenroth, H. Amor, D. Vogt, B. Schölkopf, and J. Peters. Probabilistic movement modeling for intention inference in human-robot interaction. *International Journal of Robotics Research*, 2013.

[245] A. Wilson, A. Fern, and P. Tadepalli. A bayesian approach for policy learning from trajectory preference queries. In *Advances in Neural Information Processing Systems*, 2012.

[246] C. Wu, I. Lenz, and A. Saxena. Hierarchical semantic labeling for task-relevant rgb-d perception. In *RSS*, 2014.

[247] X. Xiong and F. De la Torre. Supervised descent method and its applications to face alignment. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[248] X. Xiong and F. De la Torre. Supervised descent method for solving nonlinear least squares problems in computer vision. *arXiv preprint arXiv:1405.0601*, 2014.

[249] F. Zacharias, C. Schlette, F. Schmidt, C. Borst, J. Rossmann, and G. Hirzinger. Making planned paths look more human-like in humanoid robot manipulation planning. In *Proceedings of the International Conference on Robotics and Automation*, 2011.

[250] C. Zhang and Z. Zhang. A survey of recent advances in face detection. Technical report, Microsoft Research, 2010.

[251] N. Zhang, M. Paluri, M. A. Ranzato, T. Darrell, and L. Bourdev. Panda: Pose aligned networks for deep attribute modeling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[252] X. Zhang, P. Jiang, and F. Wang. Overtaking vehicle detection using a spatio-temporal crf. In *IVS, IEEE*, 2014.

[253] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the International Conference on Computer Vision*, 2015.

[254] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, 2008.

[255] B. D. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. Srinivasa. Planning-based prediction for pedestrians. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, 2009.

[256] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. .S. Srinivasa. Chomp: Covariant hamiltonian

optimization for motion planning. *International Journal of Robotics Research*, 32, 2013.